UNIVERSITÉ DE GRENOBLE

# THÈSE

Pour obtenir le grade de

## DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Nano-Électronique et Nano-Technologies**

Arrêté ministériel : 7 août 2006


Présentée par

**Fabien CHAIX**

Thèse dirigée par **Michael NICOLAIDIS**
et codirigée par **Nacer-Eddine ZERGAINOH**

préparée au sein du laboratoire **TIMA, Équipe ARIS**,
et de l'école doctorale **Électronique, Électrotechnique, Automatique et Traitement du Signal**


# Contributions for late CMOS many-cores processors: NoC fault-tolerant routing and auto-adaptive applications

***Contributions aux processeurs multi-cœurs massivement parallèles en technologie en rupture:***
***Routage tolérant aux fautes du réseau d'interconnexion et auto-adaptabilité des applications***

Thèse soutenue publiquement le **28 Octobre 2013**,
devant le jury composé de :

**Michel RENOVELL**, Président
Directeur de Recherche CNRS, LIRMM
**Amer BAGHDADI**, Rapporteur
Professeur, Telecom Bretagne
**Nacer-Eddine ZERGAINOH**, Examinateur
Maître de Conférences, Polytech' Grenoble
**Michael NICOLAIDIS**, Examinateur
Directeur de Recherche CNRS, TIMA

À Renée

# Foreword

First of all, I feel very grateful to Professor Renovell and Professor Baghdadi for reviewing this thesis, which required a large amount of their valuable time. Second, I would like to express my gratitude to Professor Nicolaidis and Doctor Zergainoh for offering me the opportunity of this thesis. Their flawless confidence and support have been precious on many occasions. I would like also to acknowledge the efforts of Professor Avresky who was essential on many scientific aspects. Finally, I have much respect for the administrative staff of TIMA, whose diligent efforts provide excellent working conditions to the laboratory researchers.

My thoughts also go to my family and friends who supported me for 5 -long- years. I would like to say how much their presence was essential for me not to fall into the dark side. I particularly thank my basket-ball co-players and my laboratory mates who shared many moments that I will long remember. Finally, I would like to express my gratitude to Madeleine and Alain for their proverbial hospitality.

The manuscript first features a summary in french, as required by the university. Then, english is utilized for the remaining of the thesis. First, an overall introduction presents the context and the motivations of this thesis. The second chapter presents novel fault-tolerant routing algorithms for 2D-Mesh Network on Chips. The third chapter presents techniques for the execution of parallel applications by many-core unreliable processors subject to variability. Finally, the fourth chapter presents the simulator developed during this thesis.

# Resumé

# I Introduction

## I.1 Contexte

Les nouvelles technologies ont envahi notre quotidien et de nombreux pays en dépendent sur le plan économique et social. En effet, les capacités de calcul en constante évolution ont permis l'émergence d'applications qui ont amélioré l' efficacité des agents et des moyens de communication, en plus d'ouvrir la porte à de nombreux loisirs dits numériques. Dans le futur, on peut gager que la mise au point de processeurs de calculs plus puissants sera donc nécessaire pour répondre aux besoins croissants de nos sociétés toujours plus connectées. L'utilisation de ces processeurs demandera néanmoins de pallier aux problèmes de fiabilité et de variabilité inhérents aux techniques de fabrication à venir. En effet, ces procédés produiront des circuits intégrés plus hétéroclites et plus sensibles aux conditions environnementales.

#### Évolution des processeurs

L'évolution des processeurs dans les deux prochaines décennies reste cependant très floue, et ce pour plusieurs raisons. Tout d'abord, les **demandes applicatives** évoluent rapidement et souvent de manière surprenante. Ensuite, les **contraintes physiques** donnent lieu à penser qu'un changement radical de l'industrie serait nécessaire ; pour autant, les technologues pourraient contourner les limitations physiques actuelles, comme ils l'ont fait de nombreuses fois par le passé. Enfin, **les réalités économiques** influenceront grandement les évolutions futures de l'industrie microélectronique, avec pour objectif principal le retour sur investissement et la garantie de revenus stables pour les entreprises.

L'architecture des processeurs de calcul a longtemps été dictée par les exigences et les pratiques des informaticiens qui les programmaient. Certaines applications phares telles que l'édition de texte, puis la téléphonie mobile, l'internet ou enfin les services mobiles ont beaucoup influencé l'évolution du cahier des charges des processeurs. Plus récemment, l'avènement des téléphones intelligents (*smartphones*) et de fermes de serveurs (*cloud computing*, réseaux sociaux) implique d'exécuter quasi-simultanément de nombreuses tâches ayant des contraintes de fiabilité extrêmement différentes (e.g. transactions bancaires, lecture de flux vidéo, jeux gratuits).

En ce qui concerne les contraintes physiques, on peut relever trois points principaux. Premièrement, l'energie nécessaire à chaque opération diminue avec la tension de seuil des transistors. Comme celle-ci est liée à la taille de leur canal, cette contrainte a poussé à une miniaturisation effrenée des transistors. Ainsi, dans les dernières technologies industrialisées, le canal des transistors ne mesure qu'une vingtaine de nanomètres, alors que la taille des atomes de silicium est de l'ordre du nanomètre. Deuxièmement, une partie de l'énergie nécessaire à chaque opération est dissipée sous forme de chaleur. Par

conséquent, la concentration extraordinaire de transistors qui est obtenue aujourd'hui pose des problèmes de dissipation thermique. Ainsi, de nombreux experts estiment que la dissipation thermique sera l'obstacle principal pour les futures générations de processeurs [1]. Troisièmement, la fabrication industrielle de transistors de quelques nanomètres est très difficile. En effet, les procédés industriels sont terriblement complexes et coûteux, et nécessitent des investissements énormes pour la contruction d'unités de production. Les masques de lithographie sont également tellement chers qu'il n'est rentable d'utiliser les dernières technologies que pour des puces produites en grande série. Il y a ensuite des problèmes de plus en plus aigus de **variabilité** de la production. Il est en effet impossible de fabriquer en série des puces qui soient constituées de milliards de transistors tous identiques et fonctionnels. Qui plus est, les circuits obtenus sont plus sensibles aux radiations cosmiques et aux variations thermiques et électriques. En bref, il s'agira à l'avenir pour les ingénieurs d'obtenir un maximum de performance avec une énergie limitée et des composants aux caractéristiques disparates.

Les intérêts économiques de l'industrie microélectronique influent également sur l'évolution des processeurs. En effet, les contraintes industrielles de rendement de production, de retours sous garantie et de temps de mise sur le marché modulent beaucoup l'impact des solutions techniques, car il est vital pour les entreprises de garantir le bon fonctionnement des produits vendus, vu l'impact sur leurs finances et leur image. Par conséquent, la validation quasi-exhaustive de l'architecture et le test des puces en production sont des prérequis souvent nécessaires pour l'adoption de nouvelles techniques. Afin de diminuer le nombre de puces non conformes, il est également commun de procéder à un tri en fonction de la performance de chacune, et de les vendre plus ou moins cher en fonction de celle-ci (i.e. *binning*). En dernier recours, les techniques de **tolérance aux fautes** permettent d'augmenter la fiabilité des circuits obtenus, souvent par l'ajout de circuits redondants.

L'évolution future des processeurs sera donc le fruit de contraintes et d'opportunités diverses et changeantes. On peut néanmoins noter plusieurs observations. Tout d'abord, l' **efficacité énergetique** sera une caractéristique essentielle pour répondre aux besoins grandissants de mobilité et aux problèmes de dissipation thermique. Ensuite, les processeurs intégreront de plus en plus de cœurs de calculs ; qu'il s'agisse d'une grosse dizaine de cœurs extrêmement complexes (i.e. processeurs multicoeurs), ou de milliers de cœurs exécutant des opérations très simples (i.e. processeurs massivement parallèles). Dans tous les cas, la quantité de données traitées sera plus importante qu'aujourd'hui, et nécessitera l'utilisation de circuits d'interconnection plus intelligents. Dans le cas où plus d'une centaine de cœurs seront intégrés, il faudra alors recourir aux **réseaux sur puces** ou NoC[1].

## Réseaux sur puces

Depuis les années 2000, la perspective de puces composées de dizaines de propriétés intellectuelles ou IPs[2] a posé le problème de l'extensibilité (*scalability*) de leur interconnection. En effet, l'utilisation d'un bus devient tout à fait inefficace lorsque plus d'une dizaine de maîtres y sont connectés. En se basant sur les travaux préexistants dans les réseaux d'ordinateurs, les réseaux sur puces ou NoCs ont été proposés, notamment par [2]. Les NoCs ont pour fonction de transmettre des données entre différents **nœuds de calcul** eux-mêmes, composés d'un ou plusieurs cœurs, sous forme de paquets. Un exemple de NoC est ainsi présenté à la figure 1. Chaque nœud possède son interface réseau ou NIC[3] pour le formatage des données et la gestion des transactions. Des routeurs sont disposés plus ou moins réguliérement, afin de diriger chaque paquet vers sa destination (i.e. routage). Chaque routeur embarque plusieurs **ports de sortie** et **ports d'entrée**, eux-mêmes connectés à un lien, afin de transférer physiquement les messages.

[1] Network on Chip
[2] Intellectual Propertys
[3] Network Interface Circuit

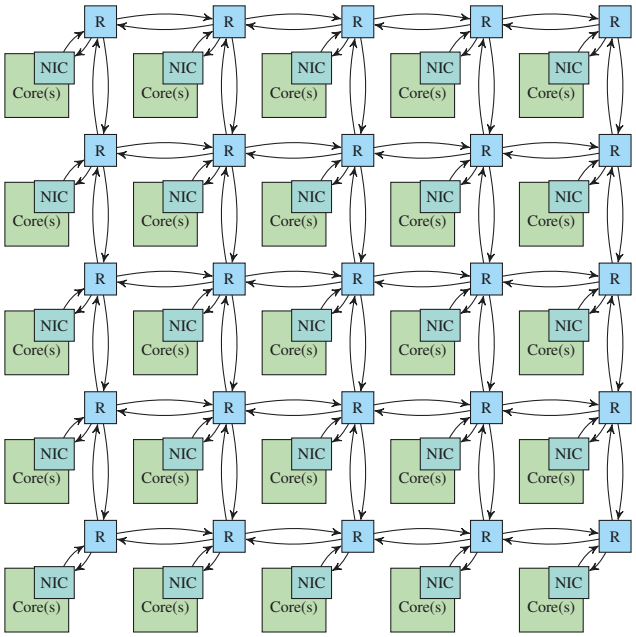



FIGURE 1 – Réseau sur puce en grille 2D

Plusieurs ressemblances existent entre les réseaux sur puce et les réseaux d'ordinateurs. Tout d'abord, le "routage par trou de ver" (*wormhole routing*) est généralement utilisé. Il s'agit de décomposer chaque message à transmettre en paquets de **flits**, et de transférer ces parties à la suite les unes des autres [3]. Un flit de tête, inséré avant le premier flit de données, contient les informations nécessaires aux routeurs pour sélectionner le chemin adéquat, et un flit de queue signale la fin du message. La segmentation de chaque message permet de réduire significativement les ressources matérielles du réseau, mais elle introduit des problèmes d'interblocage, qui compromettent son bon fonctionnement. La solution la plus commune est alors de restreindre l' algorithme de routage utilisé afin d'éviter l'apparition de "cycles de dépendances". En particulier, l'adjonction de canaux virtuels ou VCs[4] permet de simplifier la mise en oeuvre d'algorithmes de routage sans interblocage et/ou d'améliorer la performance du NoC en permettant la transmission concurrente de plusieurs messages sur un même lien.

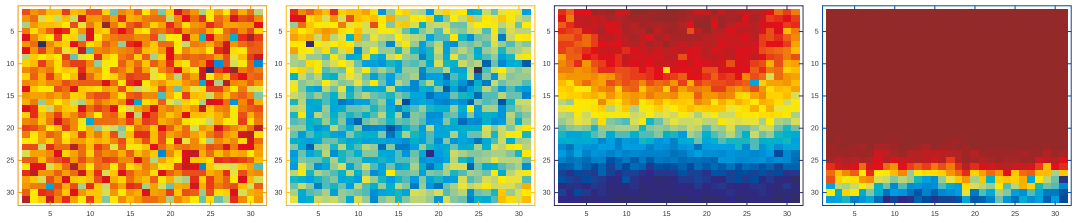
### Architecture des processeurs multicoeurs massivement parallèles

Dans cette thèse, nous considérons des processeurs multicoeurs massivement parallèles, consistant de milliers de cœurs fabriqués dans une technologie agressive, comme illustré par la Figure 2. Dans ce contexte, la variabilité et la faible fiabilité des composants du processeur sont des problèmes importants. Ces processeurs sont basés sur un réseau sur puce en grille 2D munis d'un algorithme de routage tolérant aux fautes tel que proposé dans la Section II.1, dont chaque nœud contient un ou plusieurs cœurs de calcul. La fréquence de chaque nœud est ajustée indépendamment, en fonction des besoins en puissance de calcul et des conditions de variabilité.

Dans les technologies décananométriques, il existe des variations importantes dans la tension de seuil des transistors, ce qui affecte à la fois la consommation et la performance des circuits. On peut distinguer les **variations aléatoires** et les **variations systématiques**, mais il est actuellement impossible de prévoir quelle sera leur amplitude respective, et surtout leur impact réel sur les circuits créés. On peut par contre envisager plusieurs scénarios, comme dans la Figure 2.

De plus, les fautes permanentes et transitoires menaceront l'exécution des applications, à cause de

[4]Virtual Channels



(a) 0 % (b) 6 % (c) 12 % (d) 24 %

FIGURE 2 – Fréquence maximale des nœuds de calcul dans un processeur soumis à différents niveaux de variabilité systématique

la forte variabilité des procédés d'une part, et d'une sensibilité accrue à la température et aux rayons cosmiques d'autre part. Par conséquent, les puces devront intégrer des circuits de test en ligne (BIST[5]) pour détecter l'occurence de ces fautes. Par souci de simplicité, nous considèrerons ici que lorsqu'un cœur, un routeur ou un lien devient défaillant, il cesse immédiatement d'interférer avec les autres composants (*fail-silent assumption*).

Comme proposé dans [4], les processeurs considérés intègrent une technique de saut de tension à grain fin ($V_{dd}$-*hopping*) au niveau des nœuds. Ainsi, le voltage (et donc la fréquence maximale) de chaque nœud est fixé soit à une valeur haute, soit à une valeur basse si les besoins de calcul sont plus faibles. Les nœuds inutilisés sont eux totalement désactivés (*power-gating*). Un contrôleur de nœud ajuste la fréquence en fonction des caractéristiques du nœud telles que la tension, la variabilité ou la température, afin d'éviter l'apparition de fautes temporelles.

## I.2 Motivations

Dans le cadre de cette thèse, nous considérons des processeurs composés de milliers de cœurs réalisés avec un procédé décananométrique. Une telle puce incluerait de nombreux circuits (e.g. cœurs, routeurs, liens) défectueux ou dont les caractéristiques seraient éloignées des spécifications nominales. Malgré tout, la redondance massive en cœurs de calcul offrirait une puissance de calcul exceptionnelle, en exploitant en priorité les cœurs sains les plus performants. En particulier, une adaptation dynamique et locale de la fréquence et de la tension d'alimentation DVFS[6] permettrait d'exploiter pleinement le potentiel de chaque cœur *via* la méthodologie GALS[7].

Dans ce mémoire, trois contributions sont présentées pour l'utilisation de processeurs multicoeurs massivement parallèles mais peu fiables et sujets à une variabilité importante. Premièrement, plusieurs algorithmes de routage tolérant aux fautes pour les réseaux sur puce en grille 2D sont proposés. Deuxièmement, nous décrivons une technique de gestion d'applications parallèles, tolérante aux variations et aux défaillances des cœurs de calcul. Troisièmement, le modèle de simulation VOCIS[8], développé durant cette thèse, est présenté avec une attention particulière sur les fonctionnalités qui font son originalité.

Notre approche nécessite tout d'abord de fournir au logiciel exécuté les moyens de fonctionner efficacement, en présence de défaillances multiples et de forte variabilité. En premier lieu, il est essentiel d'offrir aux cœurs embarqués un *média* de communication suffisamment fiable pour supporter l'exécution des applications cibles. Les réseaux sur puces de topologie régulière, dont la plus commune est la **grille 2D**, offrent naturellement la redondance et la simplicité nécessaires pour la tolérance aux

[5]Built-In Self-Test
[6]Dynamic Voltage Frequency Scaling
[7]Globally Asynchronous Locally Synchronous
[8]Versatile On-Chip Interconnect Simulation model

fautes des communications inter-coeurs. Dans cette thèse, plusieurs algorithmes de routage tolérant la défaillance d'une partie du NoC sont présentés, pour la topologie grille 2D. Tous garantissent l'absence d'interblocage, mais leur complexité respective leur permet de tolérer plus ou moins de défaillances de routeur et de lien. En particulier, la plus avancée est capable de transmettre les messages tant qu'il existe un chemin valide. Par la suite, diverses techniques sont proposées pour améliorer la performance du NoC en présence de défaillances, ce qui est rarement traité dans la littérature.

En se basant sur un réseau d'interconnection tolérant aux fautes, nous proposons ensuite une méthode de gestion des applications parallèles qui permet de les exécuter efficacement, malgré la défaillance et la forte variabilité de certains cœurs. Cette méthode s'appuie sur la décomposition des applications en tâches hiérarchiques suivant un graphe acyclique dirigé ou DAG[9]. La tolérance aux fautes est garantie par l'utilisation de messages de contrôle IAM[10] et la réallocation des tâches lorsque la défaillance d'un cœur exécutant est détectée. Dans un deuxième temps, le mécanisme de réallocation de tâches est adapté pour prendre en compte la variabilité et la charge des cœurs, ainsi que l'énergie dissipée dans le réseau d'interconnection, afin d'améliorer la consommation énergétique de l'application.

Afin de supporter les travaux de cette thèse, un modèle de simulation de réseau sur puces nommé VOCIS a aussi été developpé. Ce modèle C++/SystemC met l'accent sur la généricité et la réalisation de simulations de tolérance aux fautes. Ce modèle permet l'implémentation générique de nombreux aspects du réseau, et en particulier de tout agorithme de routage tolérant aux fautes. De plus, des outils d'analyse avancés et une interface 3D permettent de visualiser l'effet des différents composants, tout spécialement en présence de fautes multiples.

# II Contributions

## II.1 Réseaux sur puces tolérants aux fautes et performants

### Architecture du routeur proposé

Dans les réseaux sur puces réguliers, tels que les grilles 2D, il est possible de router les messages uniquement avec la connaissance du nœud de destination. Les algorithmes de routage proposés utilisent une liste des ports de sortie disponibles, hiérarchisés en fonction de la destination du paquet. D'autre part, l'absence d'interblocages est obtenue, comme souvent, en restreignant les possibilités de routage. Ces restrictions sont néanmoins néfastes pour la tolérance aux fautes, en ce qu'elles empêchent l'utilisation de certaines routes. Notre solution consiste donc, dans un premier temps, à diviser les ressources matérielles en 2 réseaux virtuels ou VN[11], avec des restrictions de routage différentes et complémentaires. En choisissant le réseau virtuel adéquat, il est ainsi possible de minimiser l'impact des restrictions sur la tolérance aux fautes.

Chaque port du routeur est partagé entre plusieurs canaux virtuels ou VCs. Pour chaque VC et chaque port, les messages sont accumulés dans un mémoire FIFO[12], comme illustré par la Figure 3, et le lien est multiplexé entre les différents VCs. Finalement, un allocateur de VCs contrôle leur utilisation. Pour nos algorithmes de routage, des **réseaux virtuels** ou VNs sont définis comme des groupes exclusifs de VCs à l'intérieur desquels les messages sont transmis. Dans cette thèse, nous utiliserons 4 canaux virtuels par port, 2 pour chaque réseau virtuel. Quand c'est nécessaire, une source virtuelle ou VS[13],

[9] Directed Acyclic Graph
[10] I am Alive Message
[11] Virtual Network
[12] First-In First-Out
[13] Virtual Source

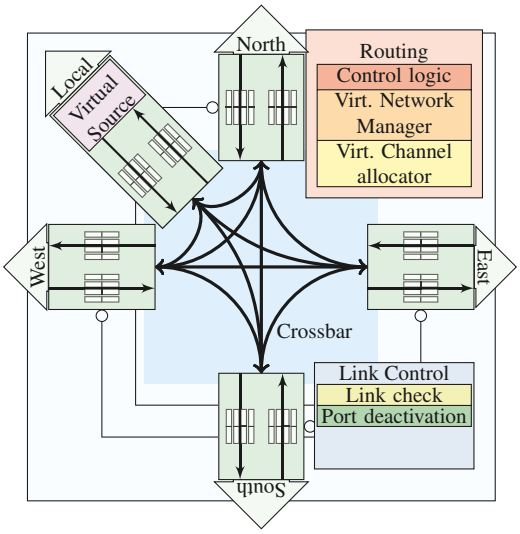



FIGURE 3 – Détails de l'architecture du routeur proposé

introduite dans la Section II.1, permet l'échange de messages entre les VNs, sans permettre l'apparition d'interblocages.

De plus, le routeur embarque un circuit de contrôle des liens, qui gère l'apparition des fautes transitoires et permanentes. En fonction de la technologie et de l'implémentation, la détection des fautes à été traitée par de nombreuses solutions, et ce travail ne prétend pas y innover. Nous considérerons donc un schéma de haut niveau compatible avec la majorité des solutions de la littérature. Un module de détection diagnostique l'état du lien en échangeant périodiquement des flits de contrôle IAM avec les nœuds voisins, et en appliquant un code correcteur sur chaque flit échangé. Lorsqu'une faute permanente est détectée, le module de désactivation inhibe tout trafic dans le port, à l'exception des flits de contrôle.

### Variante A : hiérarchie des ports de sortie et réseaux virtuels

Parmi les algorithmes de routage que nous proposons, la variante A est le plus simple. Elle est basée d'une part sur une hiérachie des ports de sortie, et d'autre part sur deux réseaux virtuels nommés *North-Last* et *South-Last*, qui assurent l'absence d'interblocages.

Dans la hiérarchie des ports de sortie présentée à la Figure 4, les directions sont ordonnées selon la relation entre le nœud destination du paquet et la position du routeur. L'algorithme de routage sélectionne le port disponible dont la direction est la plus haute dans la hiérarchie. En partant de la direction la plus haute, le routeur parcourt alors les directions par ordre décroissant de hiérarchie. Pour chaque direction, le routeur vérifie que le port existe et qu'il n'est pas défaillant. En présence de défaillances, il peut arriver qu'aucune direction ne soit disponible pour transmettre le paquet. Dans le cas de la Variante A, le paquet est alors évacué, et un acquittement négatif Nack[14] est renvoyé au nœud source.

L'absence d'interblocages est basée sur la restriction du routage dans chaque réseau virtuel. Ainsi, les ports de sorties interdits par le réseau virtuel *South-Last* (respectivement *North-Last*) sont tracés en pointillés dans la Figure 5-a (respectivement la Figure 5-b). Cette approche a été prouvée par la numérotation des liens dans un premier temps [5], néanmoins l'approche pratique proposée dans [6] consiste à utiliser une table de restrictions de direction. En effet, il suffit de parcourir une table 4x4 pour savoir si un réseau virtuel interdit une direction de sortie, comme le montre la Figure 5-c. Les

[14]Negative Acknowledgement

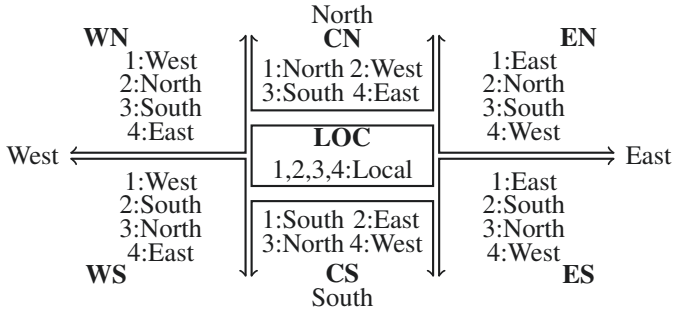
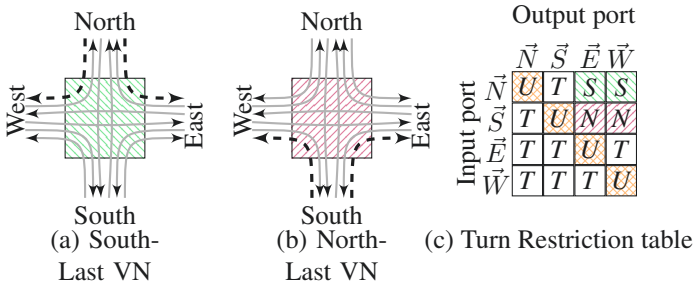



FIGURE 4 – Hiérarchie des ports de sortie




(a) South-Last VN (b) North-Last VN (c) Turn Restriction table

FIGURE 5 – Ports de sorties interdits par les réseaux virtuels *South Last* (a) et *North Last* (b)

demi-tours (marqués *U*) sont interdits dans les deux VNs. Les restrictions propres à *North Last* (respectivement *South Last*) sont notées *N* (respectivement *S*). Les autres transferts, dont ceux à destination ou provenant du port local, sont autorisés (*T*).

**Variante B : Insertion des sources virtuelles**

Dans la Variante B, afin de réduire le nombre de messages perdus, une **source virtuelle** ou VS est ajoutée à chaque routeur, pour échanger des messages entre les deux réseaux virtuels. En effet, lorsque le NoC présente des fautes de liens et/ou de routeur, le chemin emprunté par les messages peut devenir long et complexe. Lorsqu'un message ne peut être routé sans violer les restrictions, l'utilisation de la source virtuelle lui permet de continuer vers sa destination. Malheureusement, cela impose l'ajout du marquage des routeurs ou RS[15] afin d'éviter que des messages restent indéfiniment dans le réseau.

Les sources virtuelles sont utilisées pour échanger fiablement des messages entre deux réseaux virtuels, ou simplement supprimer le risque d'interblocage dans un même VN. Chaque routeur dispose d'un tampon spécifique appelé tampon VS, où les messages sont stockés intégralement avant d'être reinjectés dans le réseau, ce qui supprime toute dépendance de tampon.

Le marquage des routeurs consiste à stocker dans le message la liste des routeurs traversés, et à les éviter lors des étapes suivantes de routage. Pratiquement, il suffit de sauver la direction de chaque lien emprunté (2 à 3 bits dans une grille 2D), ce qui limite beaucoup l'impact du marquage sur la performance du réseau. Enfin, on peut noter qu'au niveau du nœud destination, le message contient la route utilisée, ce qui sera exploité pour améliorer la performance du réseau dans la Section II.1.

[15]Router Stamping

### Variante C : Addition du mode écho

Bien que la variante B aie le potentiel d'atteindre n'importe quelle destination, elle ne permet pas d'obtenir un taux nul de messages perdus. En effet, il arrive que des messages soient bloqués dans une impasse avec la Variante B, car les routeurs ont seulement une connaissance partielle du réseau. Dans ce cas, la Variante C utilise une source virtuelle pour supprimer les dépendances de tampon, et le message est ensuite routé en mode écho. Dans ce mode, le message retourne successivement dans les nœuds déjà visités, jusqu'à ce qu'une route alternative soit trouvée. Cet algorithme nous permet de répondre aux conditions les plus exigeantes, sans recourir à une table de routage. Par exemple, il est capable de gérer les cas où le réseau est presque partitionné.

### Routes explicites

Dans de nombreuses applications (e.g. applications de flux vidéo/audio), plusieurs messages sont échangés entre un même couple de nœuds source-destination. Nous avons donc introduit les routes explicites (*Explicit Path*) dans [6] pour réduire la latence moyenne des variantes B et C. Cette technique est particulièrement utile pour la variante C, lorsque de nombreux défauts entraînent l'utilisation fréquente du mode écho.

Les routes explicites sont créées dynamiquement lorsque le premier message est reçu par le nœud destination. En effet, le marquage des routeurs (RS) permet au nœud destination de reconstituer la route empruntée par le message. Dans le cas de la Variante C, les liens qui ont été re-traversés en mode écho ne sont pas inclus dans la route. La route est ensuite retournée au nœud source, qui la stocke. Lorsque de nouveaux messages doivent être envoyés à la même destination, ils sont routés en mode explicite, et suivent précisément la route mémorisée.

### Résultats expérimentaux

#### Tolérance aux fautes

Dans les Figures 6-a,b,c,d, nous montrons le pourcentage de messages perdus pour différentes tailles de réseaux, en utilisant les algorithmes A, B et C. Nous injectons des fautes permanentes de liens et de routeurs. Les résultats montrent que la variante A est suffisament tolérante aux fautes quand le taux de défaillances reste inférieur à 10 %. Lorsque les défauts sont plus nombreux, la tolérance aux fautes peut être améliorée en utilisant la Variante B. Néanmoins, certains messages peuvent être perdus avec cet algorithme, en particulier pour les réseaux les plus larges (i.e. 10x10). Dans cette éventualité, la variante C ne perd aucun message dans les réseaux non partitionnés, même pour un taux extrême de 40 % de défaillances. Il est également important de noter que les algorithmes proposés n'utilisent pas de table de routage et garantissent l'absence d'interblocages.

#### Performance

Nous avons également simulé l'utilisation de la variante C avec ou sans routes explicites pour deux tailles de réseau (10x10 et 20x20). Pour chaque configuration, nous avons simulé le transfert de 100 messages pour 50 ensembles de fautes différents. Les fautes sont injectées avant le début de la simulation et ne partitionnent pas le réseau. Nous avons considéré des ensembles de fautes de **nœud**, de **lien** ou **mixtes** (i.e. les fautes de lien et de nœud sont équiprobables).

Pour les faibles taux de défaillance, les routes explicites ont peu d'impact sur la longueur de route moyenne (Figure 7-a). Par contre, il y a une amélioration flagrante pour les taux de défaillance à partir

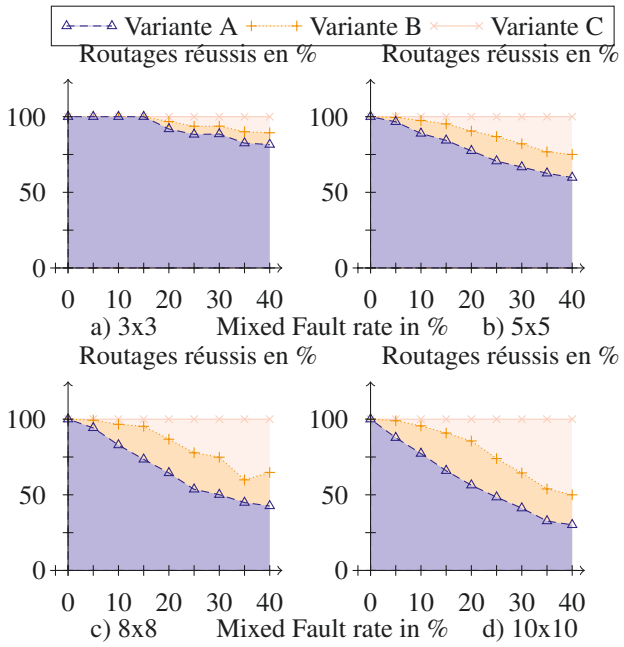



FIGURE 6 – Pourcentage de messages transmis avec les algorithmes de routage A, B et C

de 20%. Finalement, Figure 7-b montre que les routes explicites réduisent la latence de la variante C, particulièrement pour la grille 20×20.

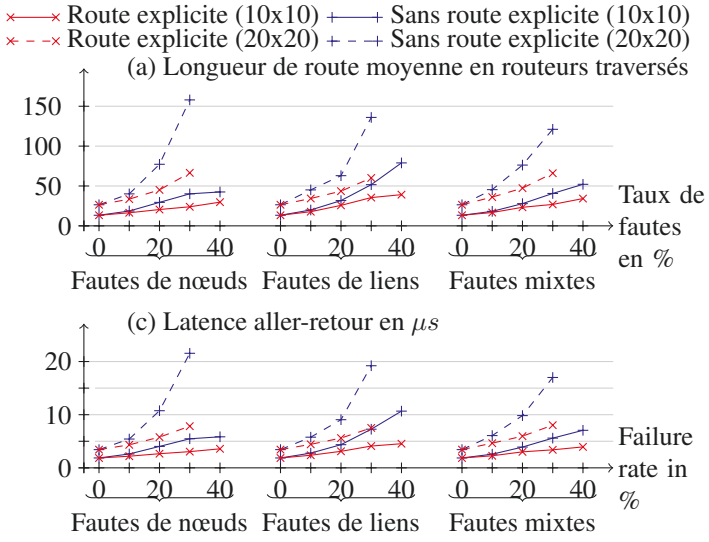



FIGURE 7 – Performance de la variante C avec ou sans route explicite en présence de défaillances

Les figures montrent que la latence moyenne tend à augmenter avec la taille du réseau et le taux de défaillance. Pour les applications de flux, qui transfèrent des messages volumineux entre quelques paires de nœuds, les routes explicites limitent l'augmentation du trafic en présence de défauts. Par conséquent, la latence moyenne serait améliorée significativement par l'emploi de routes explicites, tout en conservant l'absence d'interblocages.

## II.2 Des applications efficaces, tolérantes aux fautes et à la variabilité

La seconde contribution de cette thèse est un ensemble de techniques qui rendent les applications tolérantes aux fautes, tout en prenant en compte la variabilité présente dans les processeurs massivement

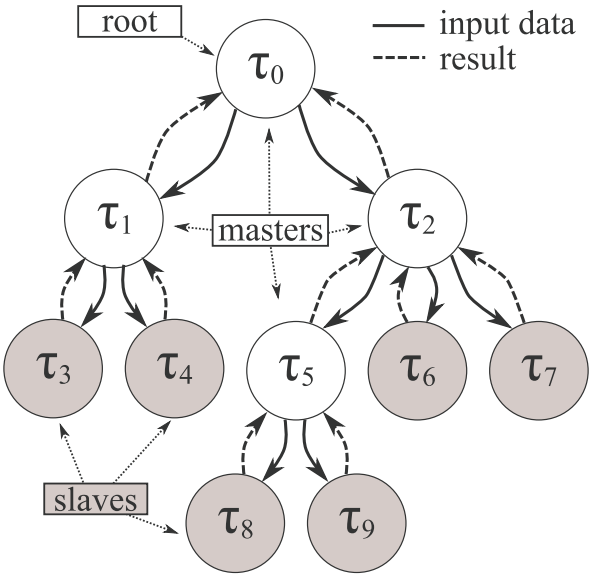



FIGURE 8 – Exemple d' application maître-esclave

multicoeurs. Notre approche permet de réduire la consommation des applications, et d'envisager l'intégration d'autres sources d'hétérogénéité.

**Modèles de programmation tolérants aux fautes**

La création d'applications parallèles tolérantes aux fautes nécessite généralement le respect de diverses règles de programmation. Dans notre cas, la technique proposée requiert l'utilisation d'un modèle de programmation de type maître-esclave.

Dans la suite, nous considèrerons une classe d'applications décrites par un graphe dirigé acyclique ou DAG[16], qui respecte le formalisme maître-esclave [7]. Les sommets du graphe représentent les tâches et les arêtes, les communications entre ces tâches. Ce formalisme est l'un des plus simples et des plus usités pour paralléliser une application en pratique. En outre, il permet de faire apparaître une hiérarchie entre les tâches, ce qui sera nécessaire pour l'approche tolérante aux fautes présentée par la suite. Généralement, une tâche maître sous-traite son travail à plusieurs tâches esclaves, puis assemble les résultats partiels. Le flot d'éxecution est inconnu *a priori* et chaque tâche peut créer de nouvelles tâches esclaves dynamiquement lors de l'exécution. Il est par contre nécessaire de connaître à l'avance une estimation du nombre d'opérations et du trafic généré pour chaque tâche. Dans ce travail, les applications sont périodiques et soumises à des contraintes temps-réel relachées (*soft real-time*), en prenant pour exemple les applications multimédia telles que le codage audio/vidéo ou le traitement d'images.

[16]Directed Acyclic Graph

## Récupération automatique en présence de fautes multiples

### Protocole de détection et de récupération

La détection des défaillances est basée sur l'hypothèse que les cœurs arrêtent toute communication lorsqu'ils deviennent défaillants. Chaque tâche envoie périodiquement un message de contrôle nommé IAM[17] à son maître. Si aucun message d'une tâche fille n'a été recu pour une période prédéfinie, elle est considérée comme défaillante, et la tâche maître réaffecte cette tâche sur un autre nœud. Lorsqu'une tâche défaillante possédait elle-même des tâches filles, elles sont également perdues, et devront donc être réaffectées.

En se basant sur le modèle de programmation proposé dans la Section II.2, la technique proposée offre donc un mécanisme simple et localisé permettant de reprendre l'exécution d'une application en présence d'une ou plusieurs défaillances de nœuds.

Le cas particulier d'une défaillance du nœud exécutant la tâche racine (*root*) est traité par la création d'une tâche clone sur un nœud différent. Avec cette technique, la tâche clone ne démarre que lorsque la tâche originale est frappée par un défaut de nœud. Elle crée alors sa propre tâche clone, et reprend l'éxecution de l'application après en avoir re-alloué les tâches.

### Affectation dynamique des tâches tolérante aux fautes

Lorsqu'une tâche esclave doit être (re-)affectée, le nœud exécutant la tâche maître lance une procédure de recherche de nœud. Il s'agit de trouver un nœud capable d'exécuter la tâche esclave, et si possible en réduisant l'énergie consommée (c.f. Section II.2). Chaque nœud étant capable d'exécuter plusieurs tâches, la recherche commence par interroger le nœud maître afin d'affecter localement la tâche fille.

En se basant sur la littérature, nous utilisons la stratégie du voisin le plus proche (*Nearest Neighbor*) pour rechercher un nœud exécutant. A l'aide de messages spécifiques, chaque nœud est interrogé sur sa capacité à exécuter la tâche fille, en partant du plus proche au plus éloigné, en évitant les nœuds défaillants ou déjà très chargés.

## Prise en compte de la variabilité du processeur

### Sélection du nœud racine

Le choix du nœud exécutant la tâche racine d'une application (i.e. nœud racine) a un impact de premier ordre sur la performance de l'application. Dans [8], nous proposons deux stratégies simples. Lorsque la variabilité systématique est faible, choisir un nœud central permet de maximiser le nombre de nœuds voisins lors de l'expansion de l'application. En revanche, si la variabilité systématique est assez forte pour faire apparaître des "régions" de nœuds plus performants comme dans Figure 2-b,c, il est préférable de choisir un nœud à l'intérieur de telles régions.

### Recherche adaptative de nœud

L'utilisation déterministe de l'algorithme de recherche de nœud aboutit à une consommation énergétique assez importante lorsque la variabilité est importante. Nous proposons donc de prendre en compte l'état du processeur lors de la recherche, afin de réduire la consommation énergétique des applications.

Un critère de recherche prenant en compte la variabilité des cœurs est donc proposé, et met en balance l'efficacité des cœurs avec leur distance vis-à-vis du nœud exécutant la tâche maître. Nous proposons également un critère d'arrêt statistique qui permet de terminer la recherche lorsqu'il devient peu probable de trouver un nœud plus éloigné qui réduise encore le critère de recherche.

[17] I am Alive Message

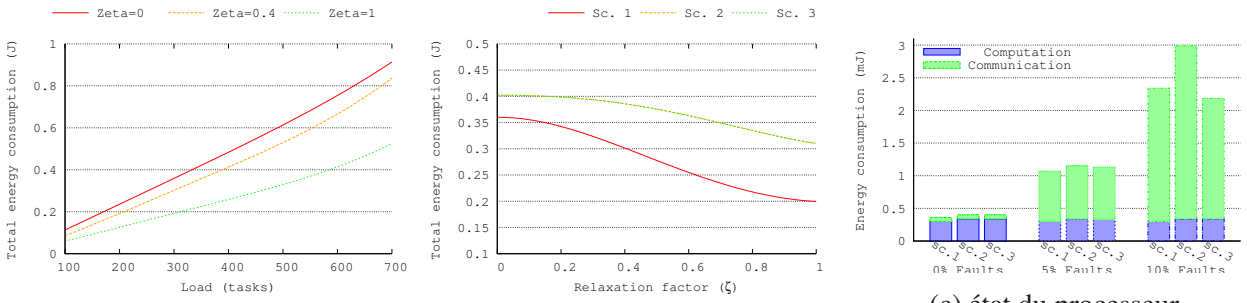
### Résultats expérimentaux

Dans ces expérimentations, on néglige l'impact de la variabilité sur le NoC[18]. De plus, nous supposons que peu de contentions surviennent dans les routeurs, et donc que la latence et l'énergie consommée pour la transmission d'un message sont proportionnelles au nombre de routeurs traversés.

La technique proposée a été simulée sur un modèle spécifique de haut niveau, où l'utilisation d'un routage tolérant aux fautes (variante A présentée dans la Section II.1) est émulée. Lors des simulations, une application de 100 à 700 tâches est déployée sur un processeur de 1024 cœurs. L'application est générée aléatoirement, graphe et caractéristiques des tâches compris, avec un ratio calcul sur communication proche de 1.

Trois scénarios de variabilité sont utilisés, afin de refléter les différentes évolutions possibles de l'industrie CMOS. Le scénario 1 correspond à une stagnation des variations systématiques et aléatoires à 6% de la valeur moyenne. En tablant sur une augmentation de la variabilité aléatoire à 24% et une annulation de la variation systématique, on obtient le scénario 2. Dans le pire des cas, le scénario 3 donnerait une variabilité aléatoire aussi élevée, avec une variabilité systématique de 12%. Lorsqu'elles sont présentes, les fautes de nœud sont injectées avant le début de la simulation.

Dans la Figure 9a, la consommation totale de l'application est donnée pour différents nombres de tâches et différents facteurs de relaxation, pour le scénario 2 et en absence de fautes. Le facteur de relaxation $\zeta$ (Zeta) modifie le critère d'arrêt de la recherche de nœuds, et offre ainsi un compromis entre l'efficacité énergétique de l'application et le coût de la recherche de nœuds en terme de communications. L'énergie consommée est réduite significativement par l'utilisation du critère adaptatif ($\zeta = 0.4, 1$) par rapport à une recherche de plus proche voisin déterministe ($\zeta = 0$).




(a) nombre de tâches (b) facteur de relaxation (c) état du processeur

FIGURE 9 – Consommation de l'application pour différentes configurations

La Figure 9b montre la relation entre consommation énergétique, variabilité et facteur de relaxation. Globalement, lorsque la variabilité et le taux de défaillance augmentent, le facteur de relaxation joue un rôle de plus en plus important. En effet, lorsque les disparités augmentent, il devient profitable de visiter des nœuds de plus en plus éloignés.

Finalement, dans la Figure 9c, l'énergie applicative détaillée est donnée, sous différents scénarios de variabilité et de fautes. La consommation globale augmente avec les fautes et la variabilité, mais notre stratégie de recherche limite cette dégradation, en prenant en compte l'energie utilisée pour les communications, et celle utilisée pour les calculs.

Cette contribution permet donc d'exécuter des applications parallèles sur des processeurs composés de centaines de cœurs peu fiables et sujets à une forte variabilité. Elle déploie les applications dynamiquement, entièrement lors de son démarrage, puis partiellement après qu'une défaillance de nœud

[18]Network on Chip

soit survenue. De plus, cette approche généraliste permet de réduire la consommation énergétique en prenant en compte la variabilité des cœurs du processeur.

## II.3 VOCIS : un modèle générique de réseaux sur puce tolérants aux fautes

Durant cette thèse, nous avons mis au point un modèle de simulation appelé VOCIS[19]. Ce logiciel avait plusieurs objectifs. Tout d'abord, il s'agissait de proposer un modèle capable de supporter la plupart des topologies et des algorithmes de routage proposés dans la littérature. Ensuite, les travaux présentés dans la Section II.1 exigeaient un simulateur capable d'injecter génériquement des fautes de routeur et de lien. Enfin, il nous fallait des moyens avancés d'analyse pour comprendre et quantifier les résultats de simulations complexes, mêlant trafic important et fautes multiples.

### Un modèle généraliste

De nombreux travaux sur les réseaux sur puce ou NoCs[20] s'appuient sur des simulateurs, soit développés en interne, soit modifiés *ad hoc*, pour obtenir les résultats expérimentaux nécessaires. Il est donc très difficile de comparer quantitativement les contributions. VOCIS offre un environnement unique pour simuler des propositions hétérogènes, et des fonctionnalités de base réduisant le temps de développement nécessaire à l'ajout d'une nouvelle technique.

#### Topologies et ports

VOCIS supporte virtuellement toutes les topologies, en les représentant par un graphe orienté dont chaque sommet correspond à un routeur ou une interface réseau. En particulier, cela inclut les réseaux assymétriques dont certains nœuds ne peuvent pas communiquer dans les deux sens.

VOCIS est également compatible avec une grande variété de ports et de liens. En effet, le modèle permet d'intégrer tout modèle de port implémentant une interface générique. Dans ce modèle, le port de sortie, le lien et le port d'entrée correspondants sont regroupés, afin de simplifier l'implémentation de divers ports et liens.

#### Algorithmes de routage et d'arbitrage

Pour les réseaux sur puce, les contraintes de complexité ont longtemps limité fortement la diversité des algorithmes employés. Néanmoins, il est admis que les futures technologies auront une plus grande capacité d'intégration, et que les problèmes de performance et de tolérance aux fautes pourront justifier l'emploi de solutions plus complexes. En particulier, les algorithmes supportant l'envoi multiple (*multicast*) ou le routage adaptatif -qui dépend de l'état du réseau- apparaîssent rapidement. On pourra également remettre en question l'utilisation d'algorithmes de routage sans interblocages, et les comparer à des algorithmes avec interblocages combinés à une technique de suppression d'interblocage. En utilisant un mécanisme générique pour l'arbitrage du *crossbar* des routeurs, VOCIS permet de simuler ces solutions sans modification de son noyau.

L'utilisation d'un graphe à la fois pour la topologie et l'arbitrage des routeurs permet d'implémenter diverses fonctionnalités avancées de manière générique, telle que la découverte de meilleurs chemins ou la détection d'interblocages, ce qui facilite la comparaison de solutions exploitant des topologies différentes.

[19] Versatile On-Chip Interconnect Simulation model
[20] Network on Chips

**Trafics**

Lors de simulations de tolérance aux fautes et de performance d'un réseau, le trafic utilisé détermine pour beaucoup les observations obtenues. Il est donc essentiel de soumettre les différentes solutions comparées à un même trafic de paquets. Dans VOCIS, chaque interface réseau (NIC[21]) ou nœud de calcul peut être adapté aux besoins de la simulation, pour modéliser un trafic applicatif ou synthétique et une gestion appropriée des requêtes.

**Prise en compte des fautes**

La simulation d'un NoC en présence de fautes nécessite plusieurs éléments. EN premier lieu, il faut sélectionner un modèle de fautes approprié et de préférence peu coûteux en puissance de calcul. Deuxièmement, l'injection des fautes doit être suffisamment générique pour faire apparaître la vaste diversité des cas de défaillance. En second lieu, il est essentiel que l'injection de fautes soit reproductible, afin de répéter le même scénario avec une solution alternative, ou simplement rejouer la même séquence lorsqu'elle mène à un résultat inattendu. En dernier lieu, plus que dans les simulations de performance pure, la simulation de fautes exige des moyens d'analyse complexes pour comprendre les effets obtenus.

**Analyse et visualisation graphique**

VOCIS a pour objectif la simulation de grands réseaux sur puce (e.g. 10x10) soumis à une variété de fautes (fautes transitoires et/ou permanentes) et à un trafic lourd et hétérogène. On peut également ajouter la complexité des algorithmes pour le routage, l'arbitrage des routeurs et la gestion des transactions. La compréhension et l'obtention de résultats numériques significatifs demandent donc l'utilisation d'outils avancés.

Premièrement, un mécanisme de traçage (*log*) évolué permet de sélectionner les messages à afficher en fonction du module émetteur et de sa gravité.

Ensuite, divers outils statistiques telle que la moyenne glissante ou la régression multilinéaire permettent de traiter des données de sortie fluctuantes. L'automatisation du traitement des données de sortie permet également de générer l'ensemble des graphiques nécessaires rapidement.

Enfin, l'ajout d'une interface graphique en 3D permet de visualiser dynamiquement le comportement du NoC et d'en obtenir des clichés. Cette fonctionnalité est particulièrement intéressante pour la bonne compréhension du comportement de réseaux soumis à un trafic important.

[21] Network Interface Circuit

# III Conclusions

Dans cette thèse, nous présentons un ensemble cohérent de techniques pour l'utilisation de processeurs futurs constitués de milliers de cœurs de calcul faillibles et sujets à une forte variabilité. Ces travaux ont été validés sur un modèle de simulation de haut-niveau nommé VOCIS, qui regroupe un ensemble de fonctionnalités uniques, permettant l'étude approfondie de réseaux dans des conditions complexes.

Plusieurs algorithmes de routage tolérants aux fautes, sans interblocages et sans tables de routage, sont décrits. Leurs différents niveaux de tolérance aux fautes permettent d'ajuster la complexité du NoC au besoin en fiabilité des applications. En particulier, la variante C garantit l'acheminement des paquets, tant qu'il existe un chemin et que le traffic est limité, et ce jusqu'à 40 % de nœud fautifs. Le débit en présence de fautes est ainsi amélioré sugnificativement. De plus, un méchanisme de routes explicites a été introduit, afin d'améliorer la performance du réseau en présence de fautes.

En se basant sur un réseau muni d'un algorithme de routage tolérant aux fautes et performant, nous élaborons ensuite une technique de gestion des applications parallèles, tolérante aux fautes et à la variabilité des cœurs de calcul. L'affectation dynamique des tâches basée sur une recherche adaptative des nœuds permet de diminuer la consommation énergétique de l'application en présence de défaillances.

# Contents

# List of Figures



[22] Virtual Networks
[23] Virtual Source

[24] Directed Acyclic Graph

# List of Tables

# Chapter 1

# Introduction

The future of the microelectronic industry has never been so unpredictable before. On one hand, stakes have rarely been so high for the advent of a new generation of processors. On the other hand, the Moore's law -which described accurately the evolution of chips performance since 1965- is severely challenged by the very properties of matter.

For half a century, the growth of modern societies relied more and more on computer science and microelectronic. In fact, the exponential increase of computing power enabled many revolutions, starting with the Apollo program to pervasive social networks and high-speed stock exchanges.

Concurrently, the microelectronic industry has been the theatre of many changes and twists due to increased physical constraints. Diverse physical limitations such as power consumption, wave propagation or reproductibility repeatedly announced the close termination of Moore's law and the stagnation of the CMOS technology. Nevertheless, high stakes pushed the microelectronic community to find innovative solutions to overcome these issues, and push this technology forward. Particularly, design and manufacturing practices evolved significantly to take advantage of the latest available technologies.

Applicative requirements also affect the evolution of computing units significantly. Initially, the sequential programming model led to processors consisting of a single -and complex- computing core. At first, complex operations were supported by the introduction of microcode. However, the increase of performance demands later pushed the introduction of deeper and deeper instructions pipelines, and the addition of branching prediction circuits for alleviating the effects of complex control flows. In the 90s, manufacturers shifted towards multi-core architectures because of power dissipation issues, which blocked the development of mobile devices. Concurrently, the increase of applications dataset size compelled processor designers to integrate ever larger memory caches.

In this techno-economic maze, we propose a set of techniques for a likely evolution of the processors architecture in the next two decades. To achieve this goal, processors would embed hundreds or even thousands of unreliable cores built with an aggressive manufacturing process. Our first contribution aims at exploiting the unreliable interconnection of cores, by providing a highly fault-tolerant deadlock-free routing algorithm. Second, we suggest a fault-tolerant method for mapping versatile applications onto the processor's cores, with respect to variability and application structure, in order to execute applications reliably and reduce their power consumption. Overall, the combination of these techniques would enable random sets of applications to practically exploit the huge computing power of these unreliable processors.

## 1.1 Trends in the applications

During the past decade, many software applications emerged and gained high popularity very quickly, such as Internet applications, GPS, and later on hand-held devices executing power-hungry media and telecom applications. While the number of applications commonly used on a day-by-day basis exploded, several trends have appeared more and more clearly.

### 1.1.1 Parallelization

Since the birth of computer science, the sequential programming model was preferred for its readability. In this model, one computation unit carries out all operations of the application one after the other. However, the parallel model emerged in order to reduce the total applications span. In this scheme, several computation units execute operations concurrently, hence reducing the amount of time required for the application to produce its results.

Unfortunately, the conversion of applications to this parallel model (i.e. parallelization) has important limitations. First, some tasks of the application cannot be parallelized, and limit seriously the speed-up obtained through parallelization ( Amdahl's law). Second, the communication between computation units may introduce a significant overhead, both in time and consumed energy. Finally, the parallelization process requires substantial development efforts, since the analysis of parallel applications' performance and reliability is more complex.

Hence, the sequential model dominated until supercomputing applications leveraged parallelization to increase the computation throughput. More recently, customer applications resorted to parallelization in order to benefit from the energy efficiency of new processors consisting of multiple cores.

In the future, the most demanding applications will likely be parallelized aggressively, since these will require increasing computation levels while the energy budget will remain stable. In this respect, the telecommunication and media standard could evolve progressively such that their implementation may be parallelized efficiently. For instance, the Mjpeg-2000 standard has a high parallelization potential, as detailed in Section 3.5.6.

### 1.1.2 Virtualization

The virtualization of applications is another recent evolution from the software industry. This mechanism consists in isolating strictly multiple applications sharing the same processor and peripherals, so that one does not affect the behaviour of others. In fact, interferences between several applications may cause the corruption of applications' internal state, following the bug of a third party application or the intent of an attacker.

In the context where more and more applications with disparate security -and reliability- requirements run on the same physical device, virtualization techniques will probably offer the application composability that is required in both mobile, workstations and server markets.

### 1.1.3 Criticity heterogeneity

Depending on the application purpose, the expected level of reliability and quality may change drastically. For instance, some applications would tolerate glitches in the results (e.g. DivX decoding), whereas time-critical applications may create life-threatening situations if outputs are corrupted or

simply miss a deadline (e.g. nuclear reactor control). Of course, providing near-absolute reliability comes at the price of huge design and manufacturing efforts, energy consumption and finally system cost.

As a consequence, designers put more or less efforts on reliability issues depending on the application target. For example, most users of MP3 players easily accept the loss or corruption of a few audio samples, whenever the device price is lower or battery lasts longer. On the opposite, spatial and military applications must guarantee adequate behaviour at all cost, under numerous degradation scenarios.

### 1.1.4 A future of handheld devices and high-performance servers

In the last decade, computers have taken a mobility shift with laptops, smartphones and lastly tablets. In these circumstances, two complementary segments emerged. On one side, handheld devices with limited power source, and wireless connectivity offering lightweight access to online services. On the other side, high performance data centers actually store and manipulate end-users data, benefiting from huge computing power, storage capacity and network bandwidth.

Interestingly, the design of both future server farms and mobile devices will focus on power efficiency. Handheld devices obviously have stringent power budget, due to limitations in the device's power dissipation, and battery life to a lesser extent. Regarding server farms, the access to power sources is not an issue and heat dissipation is accomplished through large-scale ventilation. However, the energy bill is substantial for companies providing internet services. In addition, the straightforward scaling of current designs would lead to exa-scale supercomputers - able to compute $10^{18}$ floating point operations per second- with an unreasonable energy consumption. The establishment of the Green500 list [9] reflects this trend, by highlighting the power-efficiency of supercomputers, rather than their raw performance.

## 1.2 Physical constraints and opportunities of the CMOS era

### 1.2.1 Transistor shrinking

Since the 70s, the IC[1] manufacturing technology saw the steady reduction of transistors' size as the main innovation driver. The downsizing of transistors offered higher performance and reduced the energy consumption of these devices. Though, the shrinking of transistors came with many great technological challenges. The future of the CMOS technology is threatened by fundamental physical constraints. Currently, the channel of transistors currently measures a few tens of nanometers, while the size of atoms ranges in the tenths of nanometers. At this scale, the bias of classical physic laws is significant, and the effective behaviour of transistors is readily affected by environmental parameters. As a consequence, the massive production of these devices requires complex - and costly - etching techniques.

### 1.2.2 Power density wall

While the transistors' footprint shrinked, the energy per area remained stable as long as the operation voltage could be reduced as well. However, in the latest technologies, physics have set a lower bound

[1] Integrated Circuit

to the achievable operating voltage, and the power density started rising as more and more transistors were integrated.

Hence, the performance of chips is now limited by the power that they can dissipate. In some cases, only a small portion of the chip may be used at a time, since the chip would otherwise burn quickly. As a consequence, some researchers forecast a future of chips where only 10 or 20% will be enabled at any time. Since in this projection most circuits of the chips would be powered out, this vision is named **dark silicon** [1].

### 1.2.3 Components fragility

The minute size of transistors also provokes several adverse phenomena, which impair the behaviour of manufactured chips.

#### Variability

Since devices built in decananometric technologies exhibit heterogeneous characteristics, their behaviour changes substantially from one another. In the case of transistors, the threshold voltage is affected -mostly because of RDF[2]-, which causes different power consumption and switching delay.

#### Sensitivity

These devices are also very sensitive to their environment. Variations of their temperature or operating voltage modify their behaviour significantly. In addition, since the electrical charge of cosmic rays is comparable to that of circuit's signals, a single hit may change the state of several circuit's registers or even destroy a part of the circuit.

#### Aging

In addition, the lifetime of these devices fluctuates depending on their utilization context. Indeed, a variety of aging mechanisms (e.g. NBTI[3], electromigration) degrade progressively the performance of the device, until it does not comply anymore with the chip specifications.

## 1.3 Evolutions in processor architecture

### 1.3.1 From raw performance to multi-dimensional target

Two decades ago, computational power was the only target in the processor industry, and it was reached mainly by increasing the core frequency. However, processors became more and more difficult to cool down, and new markets -such as mobile- asked for better energy efficiency.

[2]Random Dopant Fluctuation
[3]Negative Bias Temperature Instability

As a consequence, circuits dedicated to energy efficiency now entered many mainstream systems. For instance, the DVFS[4] became a state-of-art solution for personal computers processors, and in the near future power-gating and clock-gating will commonly be integrated as well.

Currently, the lifetime of processors is also stepping into the equation. In fact, the newest technologies enable the manufacturing of cheaper chips, at the expense of a reduced inherent reliability. As a consequence, designers are in the position of selecting whether additional reliability mechanisms shall be added or not. For instance, since most mobile phones become obsolete after one year, the reliability of their components is left out, in favor of reduced chip cost and development time. On the other hand, datacenters require processors with a long lifetime and mechanisms for coping with environmental effects (e.g. cosmic rays, temperature).

### 1.3.2 Cores integration

As a consequence of technological and market constraints, processors integrate more and more processing transistors. However, the exploitation of this huge integration potential does not always mean a large number of processing nodes, since novel cores' architecture also require more logical gates. For instance, the cache memory has a strong appetite for silicon and the speed-up of cores generally requires voracious optimizations.

In the future, processors will probably either embed a few very complex cores (i.e. multi-core), or consist of a larger number of cores that will feature an inferior computational capacity (i.e. many-core). In general, massively parallel processors gather several processing cores into a **processing node**, for improving the efficiency of applications. At an extreme level, MPPAs[5] will include a streaming-oriented interconnection and thousands of minimal cores designed for data processing, but with little support for control flow operations.

At the intermediate level, many-core processors will embed hundreds of autonomous processing nodes bound by a generic interconnection. In that approach, each processing node consists of one up to sixteen cores and several levels of memory caching, as detailed in Section 3.1.

In order to improve the processor efficiency for popular algorithms (e.g. cryptography or media encoding), specialized IPs[6] may also be inserted into processors. This **heterogeneity** will be obtained in a different way, depending on the type of processor. In the case of multi-core processors, these functionalities will be implemented in core extensions and/or embedded coprocessors, while massively parallel processors will insert specific cores and/or nodes amongst regular processing nodes.

### 1.3.3 Memory integration

Since the data set of applications rises steadily, processing cores require an instant access to larger and larger blocks of memory. As the quantity of on-chip memory is limited through **caching**, this technique requires important design efforts.

In the case of multi-core processors, a large amount of cache memory is allocated to each core, and a complex management is required in order to reach the full potential of the cores. On the other hand, massively parallel processors embed fewer memory on each processing node, which requires specific efforts from developpers in order not to exceed the node capacity.

[4] Dynamic Voltage Frequency Scaling
[5] Massively Parallel Processing Arrays
[6] Intellectual Propertys

In addition, the coherence of cores cache raises serious challenges for future designers. On the side of multi-core processors, the shared memory paradigm offers an unified processor where all data are available on all cores, through hardware and OS[7] mechanisms. For massively parallel processors, this scheme is hardly affordable, since the shared memory coherence does not scale easily. Rather, message passing is utilized in most designs. This approach requires that software developpers explicitly define data exchanges between application tasks, since transfers only occur upon tasks' request.

### 1.3.4 Smarter Interconnections

As processors integrate more and more processing cores, the communication between them and towards external peripherals becomes a major issue. In fact, core interconnections are a sensitive part of processors regarding performance, reliability and energy.

Firstly, the interconnection topology plays an important role for performance and reliability. While bus and point-to-point interconnections are suitable for the multi-core approach, they are not when the number of processing nodes exceeds a few tens. On the other hand, a NoCs interconnection leads to higher latency, but currently is the most scalable option.

Secondly, the user applications and the OS may rely on interconnection features to support their efficiency (e.g. hardware cache coherence, transport protocol) and reliability (e.g. fault-tolerant routing, data corruption countermeasures).

### 1.3.5 Adaptive chips

The production of processor chips is a profitable activity, yet financial risks are important in this industry. Devices consisting of billions of transistors are likely to malfunction. Hence, designers check extensively their products, based on tools provided by the EDA[8] industry. However, the uncertainties imposed by the manufacturing process do not allow the exact prediction of the chip behaviour.

As a consequence, **guardbands** are used to obtain economically viable designs. For example, the operating voltage is based on the estimation of worst-case dies of silicon, plus an additional margin for unexpected phenomena. Similarly, the operating frequency is reduced in order to maintain the integrity of worst-case dies.

Since guardbanding incurs significant degradations of performance and efficiency in the latest technologies, **adaptive techniques** emerged in the last decade. These techniques reduce the required guardbanding by matching dynamically the silicon die parameters (i.e. voltage and frequency) to its variability and aging. However these techniques set serious challenges to the EDA industry, since their validation is extremely complex. As a consequence, the processor industry is generally reluctant to use these adaptive techniques, until EDA tools provide a well-proven workflow.

[7] Operating System
[8] Electronic Design Automation

## 1.4 Contributions for unreliable many-core architectures

### 1.4.1 Global view

Amongst processor futures, this thesis considers chips consisting of hundreds of cores built in a unreliable technology. In this scheme, processors are manufactured by assembling identical tiles. Each tile usually corresponds to one processing node, which consist of a few processing cores and one NIC, plus one router and its related links. This evolution would offer several advantages:

1. Design scales with very few effort, and verification may be carried out on an individual tile only.
2. Late CMOS and more-than-CMOS technologies may be used more efficiently. Alternatives require either conservative guardbanding or low-level redundancy schemes, which reduce drastically the chip efficiency.
3. The presence of many redundant cores offers a high potential for manufacturing yield through binning. Indeed, in a processor consisting of one thousand of cores, dies could be sold -under lower performance rating- even though one hundred of these cores were unusable.
4. The support of generic applications would open many markets to the same processor chip, and improve significantly its profitability.

For instance, the *Cells* approach [10] developped at the TIMA laboratory proposes to tackle holistically the uncertainties of future manufacturing processes. In this vision, processors consisting of trillions of transistors (a.k.a. Tera-Device processors) will be hit by millions of defects. However, a multi-level framework will be able to mitigate the impact of defects with a low overhead. Overall, this *On-**C**hip S**e**l**f**-hea**l**ing Tera-Device Processor**s*** approach will offer unprecedented levels of computation efficiency, through a biologically inspired framework.

This work proposes a set of techniques required for the exploitation of unreliable many-cores processor chips, according to the *Cells* approach. First, we turn onto the core interconnect and present a highly fault-tolerant routing algorithm. This scheme supports gracefull degradation with the integration of explicit path. Second, the deployment of applications over the chip's processing nodes is addressed. This solution supports node and interconnect faults, and adapts the mapping of applications to take the best from nodes subject to substantial variations.

### 1.4.2 Achieving fault tolerance and efficiency in large NoCs

Cores interconnect has a major impact on the chip reliability and performance, especially in the context of unreliable many-cores processor architecture. In Chapter 2, a novel routing algorithm offering high fault tolerance is presented.

Firstly, our study aims at obtaining fault tolerance for 2D-Mesh NoCs. Sections 2.4.2, 2.4.3, 2.4.4 describe 3 fault-tolerant routing algorithms reflecting the thesis chronology. Each algorithm may be utilized since the most suitable depends on the reliability demand and the affordable router complexity. The latter provides the highest fault tolerance, and is able to route messages as long as a valid route exists from the source node to the destination node.

The second part is oriented towards performance in the presence of defects. In effect, maintaining high performance under faults requires additional efforts. In Section 2.5.1, explicit paths are exploited to optimize repetitive communications between the same source and destination node.

### 1.4.3 Managing multiple parallel applications in uncertain execution context

The practical exploitation of unreliable many-cores processors requires to run multiple software applications with a sufficient reliability. After guaranteeing the inter-cores communications through fault-tolerant routing, the fault-tolerant execution of applications' tasks is supported by self-adaptive application management techniques.

Firstly, an adequate programming model is described in Section 3.4. In effect, for applications to resist to multiple core faults, a hierarchical structure with explicit inter-task communications is required. In particular, this feature improves the task re-mapping efficiency and reliability.

Second, a generic fault-tolerant mapping algorithm is proposed in Section 3.5. This technique iteratively maps tasks following the application needs. When a -parent- task requests the creation of a child task, the executing node will search for a suitable node by exchanging control messages with other nodes. Ultimately, children tasks will be executed either on the same or a distinct node, while respecting the applicative constraints.

In Section 3.6, the task mapping is augmented with chip variability awareness. In effect, the generic mapping algorithm is improved to use in priority the nodes with the best characteristics. Moreover, the placement of application root tasks is discussed, and several schemes are proposed, depending on variability parameters.

### 1.4.4 Modeling the fault tolerance and performance of large interconnects

The development of these techniques demanded a flexible simulation model in order to debug and measure the impact of novel mechanisms. In Section 4, the VOCIS model is presented, with a focus on features which enable its rare versatility. This program also supports original visualization and post-processing features, which allow for in-depth comprehension of large NoCs.

# Chapter 2

# Smart routing algorithms for the high fault tolerance of large NoCs

Following the evolution of applications and processors architecture discussed in Chapter 1, the cores interconnect may soon become a bottleneck regarding scalability, chip performance and reliability. In this context, reducing the impact of cores interconnect fault, while providing high performance, is a capital issue. Fault-tolerant NoCs[1] offer an elegant solution to this challenge, and consequently received a lot of attention from both academic and industry for more than a decade now [11].

In this chapter, contributions to the field of fault-tolerant NoCs are presented. The original fault-tolerant routing algorithms discussed in Section 2.4 were initially proposed at the *IEEE International Symposium on Network Computing and Applications* in 2010 [5]. The Explicit Mode detailed in Section 2.5.1 was presented at *Design, Automation and Test in Europe Conference and Exhibition* in 2011 and was awarded the *Best Interactive Presentation Paper* [6]. Finally, an article focusing on the dynamic recovery mechanism presented in Section 2.4.1 is currently submitted to an *IEEE Transactions journal*.

After a rapid introduction on NoCs in Section 2.1, the context of this contribution will be detailed in Section 2.3. A first set of contributed routing algorithms targeting fault tolerance is presented in Section 2.4. Then, improvements of these algorithms for performance are discussed in Section 2.5. The experimental results obtained for the proposed routing algorithms are presented in Section 2.6 and algorithmic properties are shown in Section 2.7.

## 2.1 An introduction to NoCs

In this section, we give a short introduction regarding NoCs and their role in future unreliable many-cores processors. In Section 2.1.1, a brief history of NoCs is given. Then, Section 2.3 depicts the overall constitution of NoCs, and introduces the role of each components. Section 2.1.3 discusses several notions of interest in the extent of NoCs. Finally, Section 2.1.4 presents the causes, implications and state-of-the-art solutions to the NoCs unreliability.

[1] Network on Chips

### 2.1.1 History

Chronologically, the oldest ancestor of NoCs dates back to the 1960's and the beginning of computer networks. In this epoch, computers situated in different cities were communicating to each other through phone wires. However, whereas phone communications used circuit switching techniques, where the complete set of required wires is granted for the duration of a phone call, the computer networks quickly adopted packet switching, where packets of dataonly own wires for the duration of a transfer from a router to another.

Since the 1980's, the creation of computing centers -with increasing performance requirements- posed new challenges to the networking community. During this decade, elementary elements of contemporary NoCs were introduced in the field of computer networks (e.g. Mesh topology, crossbar arbitration).

In the 1990's, the democratization of computing centers to companies and academias coupled with new chips capacities paved the way to more complex networking techniques, such as modern fault-tolerant routing algorithms and VCs[2].

NoCs finally made their debut in the 2000's [12, 2]. The apparition of this paradigm is due to the increase in chips integration capacity. In effect, the communication issue came with the apparition of SoCs[3] aggregating multiple IPs[4] in the same chip. While a bus (Figure 2.1-a) could easily handle communications among a few IPs, the arbitration latencies render this communication media inpractical beyound a dizain of IPs. In effect, in the context where chips consist of dizains of interconnected IPs, the typical use cases often require that several pairs of IPs communicate simultaneously. Conversely, crossbars (Figure 2.1-b) offer point-to-point connection to processing nodes, but the number of long wires rises quadraticly. Hence, a more complex interconnect technique was necessary to provide the required performance and power efficiency.

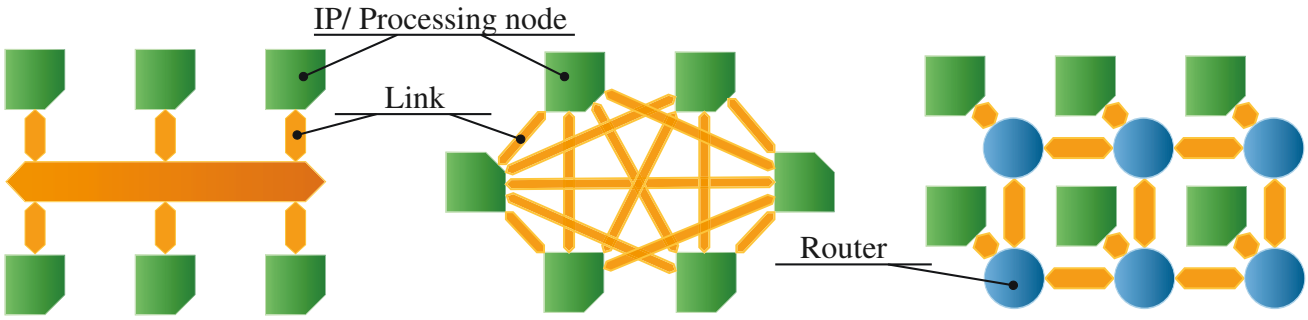



Figure 2.1: Widespread core interconnect types

In comparison, NoCs enable concurrent communications, and therefore offer a scalable and generic interconnect solution. However, while NoCs inherit most techniques from their computer network ancestors, the IC[5] domain does have unique constraints, most noticeably the silicon area and power efficiency. Hence, designers had to rethink most aspects of NoCs and eventually propose significant modifications. As a consequence, the concept of NoC emerged during the last decade and huge academic effort led to an abundant literature on the subject [13]. Yet, the industry was slower to adopt this technology. Currently, NoCs embedded in commercial products are either tightly customized in the context of hand-held devices, or part of niche-market processors for HPC[6], such as Tilera's TILE64

[2]Virtual Channels
[3]System on Chips
[4]Intellectual Propertys
[5]Integrated Circuit
[6]High Performance Computing

[14], Kalray's Multi-Purpose Processor Array [15] or Adapteva's Epiphany IP [16].

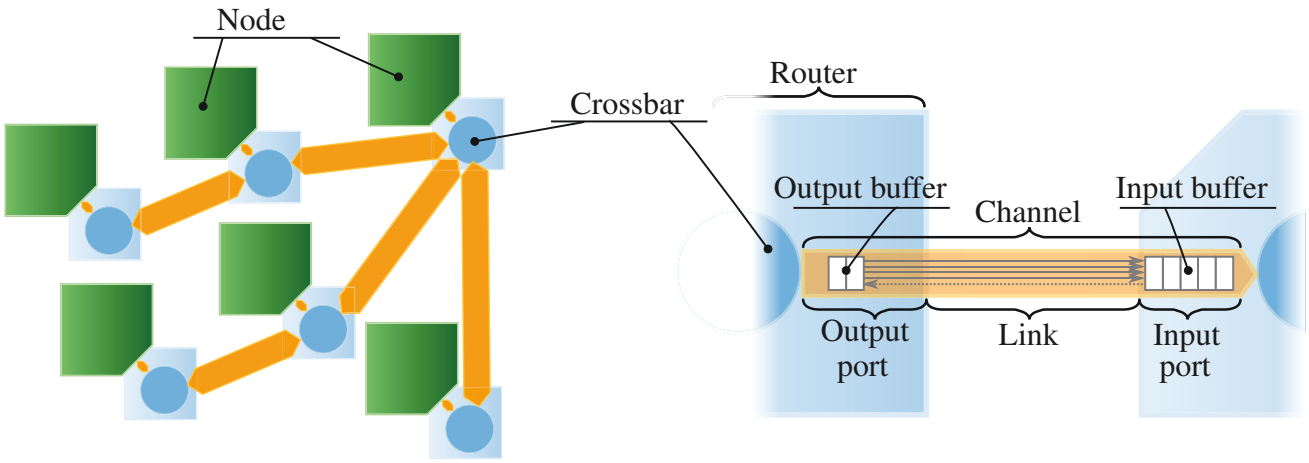
Since NoCs are critical for future processors, alternative manufacturing and design options are also investigated extensively. Outside of standard CMOS technology, two outsider techniques have been proposed by researchers. In photonic networks, light would transport data with near-zero energy, the manufacturing of waveguides and resonators together and the relative inefficiency for low traffic still pose limits to this approach. Alternatively, Radio Frequency networks would use integrated antennas to communicate data both inside and around the die [17, 18].

### 2.1.2 Constitution

In this section, the NoC structure is summarized. The role and constitution of NoCs' routers, links and NICs[7] are discussed respectively in Section 2.1.2, 2.1.2, 2.1.2. Followingly, NoC topologies are introduced and common examples are given in Section 2.1.2. Following Figure 2.2-a, a typical NoC is composed of a set of **nodes**, which embed the required peripherals. Each node may contain specific IPs or several processing cores as in Chapter 3. One NIC is present on each node to convert messages into NoC packets, as discussed in Section 2.1.2. The NoC also includes a set of routers, which direct packets to their destination. As detailed in Section 2.1.2, a router consists of a crossbar and several **ports** storing incoming and outgoing flits Depending on the topology, **Links** are inserted to bound routers together.

In Figure 2.2-b, the structure of NoC's channels is detailed. Each unidirectional channel regroups one output port, one link and one input port. Ports essentially consist of buffers for the storage of in-transit data. In this thesis, the associated link, channel and buffers are used interchangeably, when the context does not suffer from ambiguity.

In this thesis, the representation of interconnects will generally symbolize routers with their crossbar and channels will stand for the associated output port, link and input port.




a) A planar irregular NoC

b) Detail of the structure of a NoC channel

Figure 2.2: An example mesh with routers, channels and nodes

[7] Network Interface Circuits

### Routers

In packet switching networks such as NoCs, routers are required between each link. In effect, packets are transmitted iteratively from the source node to the destination node, following a sequence of links. NoCs routers generally consist of buffers for storing incoming and outgoing packets, a crossbar for transmitting incoming packets from their buffer to the selected output link, and a specific logic for controlling the router operation. Several publications yet replace the generic crossbar by a partial set of port connections, for the sake for silicon area [19].

The primary role of routers is to transmit received packets to an appropriate output channel. The action of selecting which channel shall be used by which packets is called **routing**. Following the schematic pipeline proposed in Figure 2.3, two aditional computations are required for a packet to traverse the router. The **output channel allocation** is required for the wormhole switching -discussed in Section 2.1.3- to ensure that no packet sneaks into another, and that payload flits follow the head flit. In the case of VC switching also discussed in Section 2.1.3, VCs are allocated during this step in a similar way. Finally, the **crossbar arbitration** grants timely access of the crossbar wire to flits in a non-conflicting manner. In effect, several packets may request simultaneously the same output port, and cause a conflict. As a consequence, the router has to ensure that only one packet at the time uses a given output channel and its crossbar signal. Unfortunately, this functionality requires some attention in order to avoid the so-called packet starvation. In effect, naive implementations may allow situations where a packet would wait infinitely for a crossbar grant. On the performance side, the arbitration policy may be determined following considerations such as QoS[8] or congestion-awareness.

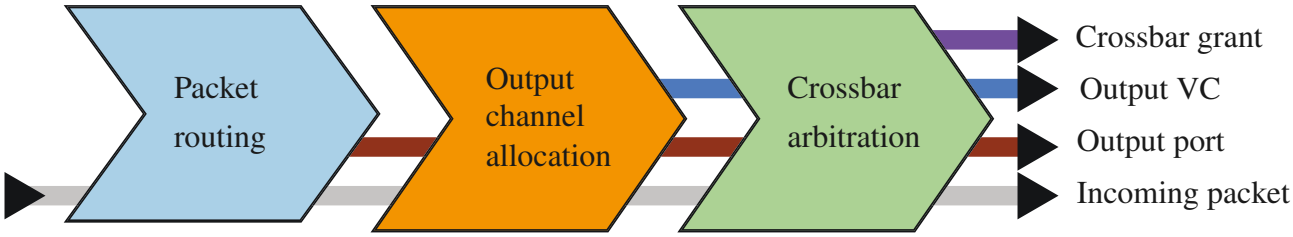



Figure 2.3: A schematic view of the router pipeline

The packet routing is the most important of the router features. Based on the packet header information and local information (e.g. routing table, neighbors state), an output channel is selected following a predefined algorithm. Since the algorithm is tied to several NoC design parameters and applicative objectives, there exist many different routing algorithms. However, a few are presented in the state-of-art of Section 2.2.

Delays in the router pipeline described in Figure 2.3 generally force designers to create a -partial- storage of the packets at routers ingress (i.e. input buffers). Upon egress, additional buffers may be required (i.e. output buffers) to support link flow control and/or VCs. Overall, the size of these buffers mainly depends on the target traffic characteristics (packet size, pattern), the link flow control and the latency of the router's pipeline. Since buffers are responsible for a significant part of the power consumption, many researchs offered to reduce their size while maintaining a comparable level of performance. For instance, several publications propose to share these buffers between router's ports [20] at the cost of few additional wires. Finally, deflection routing proposes to shift radically the router pipeline with the aim of reducing the router buffers to the strict minimum [21, 22, 23].

[8]Quality of Service

### Links

The physical transportation of NoC packets is achieved through links, following Figure 2.3. While naive designs essentially feature wires, actual designs also include quantities of control and signalling circuits. First of all, additional signals are required for the control of the transfer of data (i.e. flow control), and optionally for information regarding congestion or defects. In fact, the flow control is generally the main challenge regarding links design. In effect, the buffer beyond the link may be full already, and packet would have either to be emitted later or be re-emitted until there is room. Hence, router buffers need to be dimensionned according to the flow control strategy, in order to minimize its impact on the interconnect performance.

Another pitfall stems from the physical conception of links. In effect, links often include repeaters to reshape and optionally re-synchronize signals, since they suffer from propagation delays and electromagnetic perturbations. Of course, the insertion of repeaters exacerbates the flow control issue presented above.

### NICs

NICs are used to interface the NoC with its nodes. In effect, the transmission of messages requires several transformations before injection to the interconnect and after arrival to the destination. Moreover, NICs are responsible for implementing functionalities demanded by the transport-layer protocol.

The principal role of NICs is the serialization of messages. In a first step, the message may be splitted in different packets, which size is limited following the interconnect internal design. Most NoCs support only packets which payload amount stays under a given MTS[9]. In the wormhole switching explained in Section 2.1.3, each packet is further reduced into flits of constant size (usually 32 or 64 bits) and two additional flits are appended for ETE[10] flow control . In Figure 2.4, a 10-words message is split into 3 packets, each of them being further divided in flits of 1 word (4 bytes). At this stage, Head and Tail flits are appended to each packet.

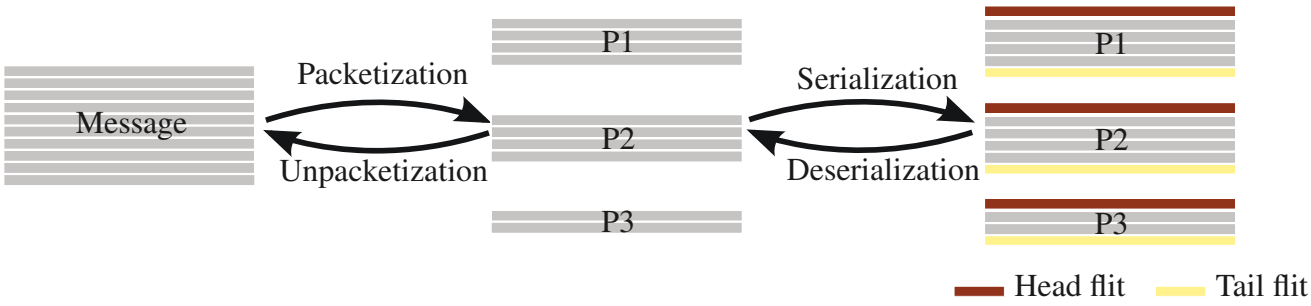



Figure 2.4: An example of conversion process from message to flits

As for computer networks, transport-layer protocols tightly depend on the targeted application model. As a consequence, there is a large variety of proprietary and open protocols. In the shared memory application model, NICs natively supports read and write transactions, i.e. one node may read or write to the local memory of another node. This approach relieves developpers from memory hassles, but poses performance issues and demands complex NIC designs. Alternatively, message passing architectures do not have these requirements, but leave applicative developpers with the burden of specifying data exchanges explicitly.

[9] Maximum Transmission Size
[10] End-To-End

There is also an extensive set of additionnal functionalities improving performance and/or reliability. In fault-oblivious schemes, most common features are outstanding requests support and packet reordering. The support of outstanding requests allows that multiple packets transmission occur simultaneously, thus improving the interconnect throughput and latency. In addition, the packetization of messages incurs that NICs support packets reordering. In effect, packets may arrive at destination out-of-order due to the routing adaptiveness, as introduced in Section 2.1.3. In addition, researchers such as [24, 25] have proposed that NICs regulate the traffic in the interconnect though congestion-aware features. However this approach requires an intimate knowledge of the complete application and interconnect behaviour. Reliability features will be discussed in Section 2.1.4.

### Topology

In any network, the positioning of routers and links is denominated as topology. Small-size interconnects can settle with irregular topologies, where there is not simple relation between nodes and their position. However, as interconnects scale up, the topology regularity becomes mandatory to mitigate the effects of scaling. Hence, this thesis focuses on regular topologies, especially the widespread 2D-Mesh. However, other topologies are discussed here, since different applicative and manufacturing contraints may drive the shift to different topologies. For instance, the introduction of novel packaging opportunities such as TSVs[11] or 2.5D currently provokes a growing interest for the 3D-Mesh topologies.

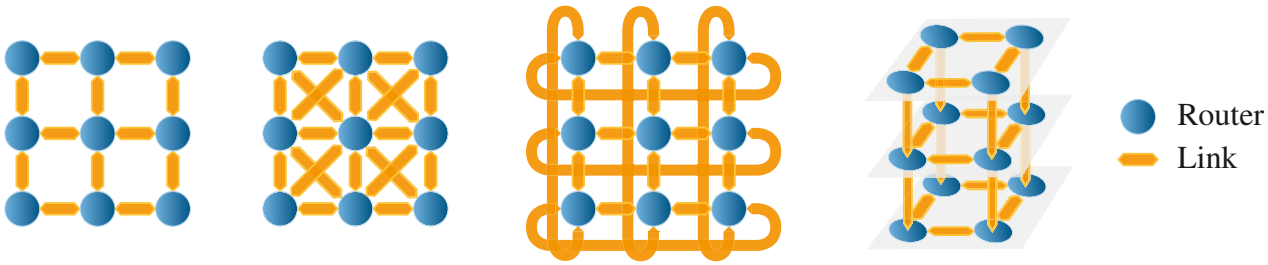
In a nutshell, topologies are characterized by their diameter (i.e the maximum distance between any pair of nodes in the network) and their average distance (i.e the average distance between each two nodes of the network). Topology distance is counted in **hops**, which corresponds to the traversal of a link from one router to another. The notion of neighbor nodes is defined followingly, as the set of nodes which respective routers are distant by exactly one hop.

The rough comparison of topologies is generally based on the tradeoff between the average distance observed by the application, and the topology overhead, which depends on the number of links (i.e. wires) and the their physical length (c.f. Section 2.1.2). Basically, the reduction of the interconnect diameter takes longer and/or more wires. The reduction of average distance demands a lower overhead, and leads to hierarchical topologies with a lower genericity regarding the application.

In the context of low-power specific SoCs such as mobile phones' processors, designers generally prefer *ad hoc* interconnects generated based on the application(s) communications graph [26, 27]. In this case, the obtained topology is usually denominated as planar irregular. [28] even proposes to configure the interconnect topology at run-time, depending on the application requirements.

In the field of NoC regular topologies, mesh-based topologies focus the vast majority of academic and companies efforts. In effect, meshes offer tractable routing functions and readable representation for low dimensionality meshes. In Figure 2.5, a few mesh-based topology variants are depicted. The 2D-Mesh is the most common topology to the extent of NoCs. Note that this topology leads naturally to the concept of node rows, columns and cardinal points. The 3D-Mesh can be seen as a stack of 2D-Meshes, and gain momemtum as die stacking techniques reach maturity. A classical variation of meshes is torus, which bounds opposite routers together as shown in Figure 2.5-c. Finally, several publications propose a variation of 2D mesh with diagonal links [29]. The D-Mesh topology enables higher redundancy and an higher level of adaptivity from lower complexity routing functions. However, the addition of diagonal links either creates heterogeneity amongst links or reduces the maximum frequency of the whole interconnect. Finally, the degraded 2D-Mesh (i.e. a 2D-mesh which has lost some of its routers) is of common use for fault-tolerant routing algorithms.

[11] Through Silicon Vias




a) 2D-Mesh b) D-Mesh c) 2D-Torus d) 3D-Mesh

Figure 2.5: Commonly used mesh-based topologies

Rings is another prolific topological family. The pure ring topology is rarely used outside of photonic network propositions, due to its high diameter and average distance. However, improvements such as STM's Spidergon offer low wire count and reasonable characteristics through the addition of cross links. The Quarc NoC presented in [30] further enhances the Spidergon topology with bi-directionnal cross links, in order to support multicast and broadcast communications.

### 2.1.3 Elementary notions

NoCs are a complex matter, with a large number of theories and associated taxonomies. In this section, a restricted set of concepts is presented to the reader, to ensure the comprehension of contributions presented in further sections. As a consequence, only superficial explanations are given here, but the interested reader will find all missing details in [3, 31].

#### Packet-switching techniques

Classical packet-switching networks are able to carry many packets with limited physical resources. Yet, alternative schemes even improve the network efficiency. For instance, each packet may be further divided into flits thus reducing hardware requirements. The VCT[12], wormhole and VC switching paradigms are all based on this approach, but exhibit some differences of interest.

In VCT switching, packets are splitted into flits at the source NIC and each flit is transfered sequentially over the traversed links and crossbars, whereas packet switching would transfer the whole packet at once. Yet, the routing of a packet requires that **every** flits be stored in the input buffer of each router. In the context of NoCs, the main issue with VCT is that each router's input buffer has to fit an entire packet, even though some researchers reported that efficient VCT designs may exist too [32].

Wormhole switching too divides packet data in **payload flits** of equal size. However, 2 distinct flits are added for ETE flow control. One **head flit** containing the routing information is inserted upfront of the payload flits Finally, a **tail flit** is added after payload flits. During the traversal of each router, the head flit is used to decide of the next routing step (i.e. the next link to traverse). Following packet routing, utilized channels are locked for the exclusive use of the packet. Forecoming flits of the packet follow the same route, and the tail flit signals that the channel exclusivity is no longer required, thus it may be used for another packet. As a consequence, the packet does not wait for all flits at each router traversal, and may spread over multiple routers. This scheme quickly became mainstream for NoCs because it generally allows smaller latency and router buffers, that is, in the integrated world, less power consumption and less silicon area.

[12] Virtual Cut Through

The VC switching is another product from the computer network community, based on the wormhole switching approach. In effect, since the progression of packet flits is affected by the arbitration decisions of other routers, channels may be owned by a packet but no flit may be transfered by the associated link. Wormhole routing would leave other packets waiting for the very same link because of channel contention. VC switching alleviates greatly this issue, by multiplexing links between several VCs, such that several packets may traverse the same physical link almost simultaneously. This scheme incurs the addition of separate buffers for each VC and thus it is often considered as costly by designers, but many academic and corporate researchers found that the performance increase justified this overhead in many situations.

### Redundancy, adaptivity and guarantees

In a global perspective, the redundancy of a resource offers the capacity to create a system that will opportunistically prefer the most suitable instance. In this approach, designers hope that if the system adapts to its current state, the overall performance will improve. Indeed, a well-designed system working under typical load should exhibit higher average performance. However, as system complexity grows and working conditions vary, adaptivity may lead to sporadic degradations. As a consequence, the system adaptivity may negatively affect observable guarantees. In essence, while adaptivity improves average, it degrades extrema behavior.

For example, most NoCs transfer packets as soon as the requested resource is available, thus improving the average behavior. An alternative approach is proposed with the time-multiplexed NoCs, such as Aethereal [33, 34]. In that scheme, packets are transfered on predefined slots of time, thus reducing the arbitration effects, and effectively offering Guaranteed Service to real-time constrained applications. A Best Effort service is also available, but the average performance is lower than mainstream NoCs, unless time slots are carefully optimized for the application(s) at use.

A second example relates to the routing of packets itself. In effect, each packet may use a large variety of routes to reach its destination, as illustrated by Figure 2.6. Black route is obtained by following the XY strategy. Blue route follows the diagonal from $S$ to $T$. The red route shows a non-minimal strategy, i.e. packet will do 9 hops instead of the minimum, 7. For a 2D-Mesh interconnect, there is indeed a vast space of routing functions. When a designer selects a deterministic routing such as XY (in black), any message from node $S$ to node $T$ will follow the exact same sequence of routers and links. Based on this information, a designer may devise guarantees upon the message latency, if the application(s) communication graph is known too. However, the obtained latency may be higher than an adaptive routing algorithm which selects the output channel based on the next hop routers state (in blue). Even more, a non-minimal routing algorithm such as Variant B (in red) may increase the number of traversed links in order to avoid some congested or faulty routers. In that case, the worst-case number of hops for a message is excessively high (24 hops) but this routing algorithm is able to improve the average performance and resiliency of the interconnect in the presence of faults.

### Routing deadlocks and livelocks

In the field of software and networks, problems related to deadlocks appear frequently, under versatile and sometimes unexpected forms. A deadlock is, generally speaking, a situation where two or more entities are unable to progress because each is waiting for another to finish an action.

While wormhole routing is well suited for NoCs, it poses serious deadlock issues. In effect, each packet reserves a serie of router channels as it goes, thus creating **dependencies** from one router's channels to another router's channels. Therefore, a deadlock-oblivious routing algorithm may create

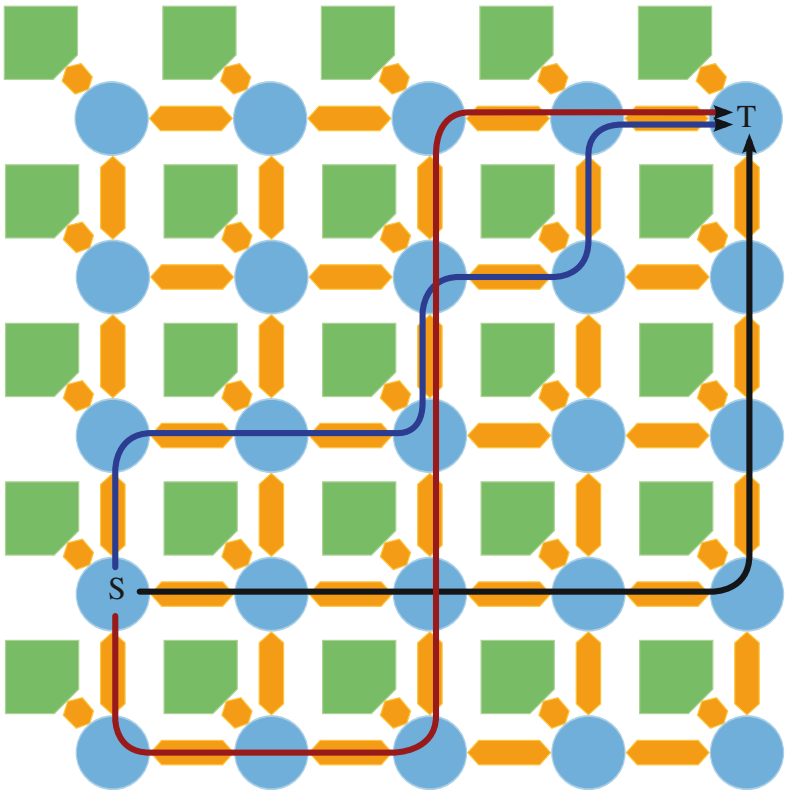



Figure 2.6: A few possible routes from node $S$ to node $T$

a cyclic channel dependency. Most routing algorithms therefore avoid the creation of deadlocks by restricting choices or splitting the interconnect traffic onto different isolated resources (e.g. VCs, time multiplexing). There are a few theories on deadlock-free routing, of which Dally's [35] and Duato's [36] are probably the most widespread. For mesh topologies, the turn model discussed in [37] led to most of the current routing algorithms. There are also some valuable inputs from the formal verification community [38, 39], to verify deadlock freedom and reduce the deadlock avoidance overhead. Alternatively, some approaches propose to permit few deadlocks to arise, but provide means to detect and recover from them [40, 41].

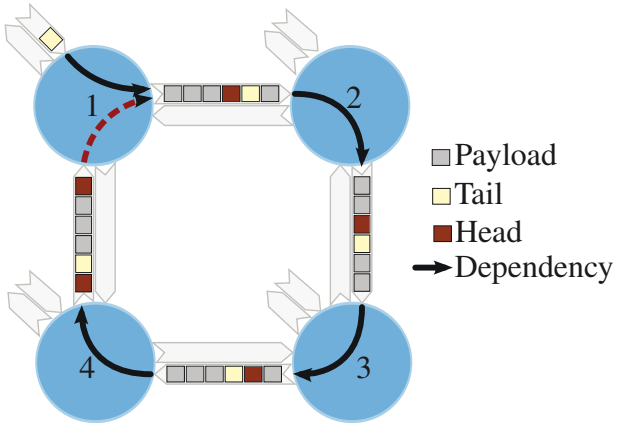



Figure 2.7: A minimal example of wormhole deadlock

In Figure 2.7, an example of wormhole switching deadlock is displayed. Each node connected to router $1, 2, 3, 4$ sends packets to the opposite node, following the clockwise direction. After a few cycles, all packets are in-transit, but blocked forever. In effect, the packet incoming at crossbar $1$ from $4$ requests channel to node $2$. Deadlock might be resolved by removing one of these packets, but it would not even have existed if another routing strategy had been chosen (e.g. XY).

**Definition 2.1.1** (Deadlocks)**.** *Given the set of packets $\mathcal{P}_t$ in the interconnect at time $t$, $\mathrm{Channels}(p_i)$ the vector of channels locked for packet $p_i$ and $\mathrm{Next}(p_i)$ the next channel requested by packet $p_i$ following the last routing decision; a deadlock occurs when there exist a subset $\{p_1, p_2, .., p_k\}$ of $\mathcal{P}_t$ such that for each packet $p_{i,1\leqslant i\leqslant k}$, $\exists p_{j,1\leqslant j\leqslant k} \neq p_i$ such that $\mathrm{Next}(p_i) \in \mathrm{Channels}(p_j)$.*

Unfortunately, the deadlock freedom constraint is directly conflicting with the adaptivity property discussed in Section 2.1.3. In effect, deadlock freedom is generally obtained by reducing the number of legal routing options, while adaptivity builds on the combination of as many routing options as possible. As a consequence, cumulating both deadlock freedom and adaptivity does not go without a significant growth of the routing complexity.

**Definition 2.1.2** (Livelocks)**.** *Let us consider sets of packets $\mathcal{P}_t$ and $\mathcal{P}_{t+T}$ in the interconnect, respectively at time $t$ and $t+T$. A livelock occurs when the same packet $p$ is present both in $\mathcal{P}_t$ and $\mathcal{P}_{t+T}$, while $T \rightarrow +\infty$.*

The livelock freedom is also incurring additional overhead for adaptive routing algorithms. In effect, packets may keep travelling in the interconnect infinitely, following local adaptive decisions. The mainstream solution is to use a predefined TTL[13] counter, which is decreased at each packet hop. When the TTL reaches 0, the packet is drained.

### 2.1.4 NoCs in unreliable conditions

In this chapter, we focus on fault-tolerant routing algorithms. As a consequence, a rapid background on NoC faults is given is this section. A few NoC defect modes are listed first. Then, some state-of-the-art detection mechanisms are described in Section 2.1.4, and NoC fault tolerance techniques are described in Section 2.1.4.

#### Defect modes

From a theoretical perspective, NoC defects are usually categorized following the affected circuit (i.e. router, link or node) and the duration (i.e. transient, intermittent or permanent). A transient fault occurs and disappears instantly, while an intermittent fault remains long enough for the system to discover it before it ends.

From a pratical point of view, a large number of physical effects may provoke a fault. In the current state of technology, the principal causes for faults are cosmic rays, variability and aging, as described in the introduction Section 1.2.3. For instance, [42] proposes an analysis of the effects of manufacturing variability on the maximum usable frequency of links. Regarding the complete interconnect, [43] proposes Static Timing Analysis to detect gate paths which are likely to fail, depending on temperature. Both works suggest that the NoC variability causes intermittent and permanent router, link and node faults. In effect, forecoming technologies are subject to complex variability effects such as NBTI[14] and progressive aging [44]. Regarding transient faults, the discharge of cosmic rays is the main contributor along with crosstalk. In both cases, the circuit is still functional, but the state of one or more bits is affected.

Finally, all faults do not affect the interconnect functionality equally. In effect, the defect of a router renders attached links unusables and *vice versa*. Moreover, when a set of faults partitions the interconnect, the routing algorithm is unable to transmit packets across the border.

[13] Time To Live

[14] Negative Bias Temperature Instability

### Fault detection

Depending on the technology and implementation, many solutions have been proposed for the detection of faults. Permanent and intermittent faults are generally detected through periodical BIST[15]. When a fault is detected, the BIST circuit is responsible for deactivating the tested circuit. Hence, additional links and nodes faults are diagnosed during run-time, using neighbour mutual test. Either BIST circuits are capable of emitting a signal to associated resources, or an IAP[16] scheme is used. In that case, each router sends periodically an IAP to each of its neighbours. When a router does not receive any IAP for a too long period, it considers the link as defective, and inhibits the all traffic to this port, except periodical IAP, which will allow the recovery of the link in the case of an intermittent fault.

Regarding transient faults, logic circuits (e.g. routers and NICs) are protected through double latch circuits such as GRAAL [45], which detect (and corrects) timing faults caused by variability and bit flips caused by cosmic rays and crosstalk. The protection of links is based on the addition of ECC[17] to packets. Fault detection either occurs at router level for HBH[18] schemes or upon packet reception for ETE schemes.

### Fault-tolerant architectures

In the state-of-the-art, there exist several complementary NoC fault tolerance techniques, applying to different circuits of the NoC

#### Routers

Fault-tolerant routers design minimaly requires the protection of the control logic, but many researchers also proposed to cope with crossbar and buffers defects. In [46], the Vicis router is proposed. This architecture is able to cope with multiple permanent and transient faults. In [47], a lightweight fault-tolerant router is proposed to cope with permanent faults. In this approach faulty routers are bypassed by enabling a direct connection from a port of choice to the NIC upon router's fault. In these schemes, the router is still able to route (some) messages and maintain connectivity of the attached node, despite the occurence of faults within its logic.

The routing algorithm is another major contributor to routers fault tolerance. In effect, the routing algorithm may bypass faulty routers (and avoid packets loss) or even select fault-free output channels in a partially defective router. The field of fault tolerant routing algorithms will be discussed longer in Section 2.2.

#### Links

In the presence of transient faults, a flit may be corrupted while traversing a link. In that case, HBH retransmissions may be used. In that scheme, a copy of flits is stored in the output buffer and retransmitted until the downstream router acknowledges it.

Alternatively, the link may use FEC[19] by transmitting flits augmented with enough redundancy to allow their reconstruction in the downstream router. For instance, [48] surveys link protection techniques based on ECCs, and proposes the utilization of Orthogonal Latin Square Code.

Regarding permanent faults, researchers have proposed different schemes to utilize the remaining link wires. In [49], links consist of two bi-directional channels, so that when one channel fails, the

[15] Built-In Self-Test
[16] I am Alive Packet
[17] Error Correcting Code
[18] Hop-By-Hop
[19] Forward Error Correction

other is able to sustain the link functionality. In [50, 51], each link's channel allows different levels of serialization to adapt to wires defects. This approach is particularly suitable for high speed links with frequent defects such as TSVs.

**NICs**

Alternatively to HBH fault-tolerance schemes, ETE approach only provides protection means in NICs A composite approach is of course possible, if not enviable [41, 52]. In effect, ETE schemes roughly offer a low-cost protection while HBH techniques provide a low latency correction.

ETE schemes are based on packets Ack[20] and retransmission. The NIC keeps a copy of each sent packet, until a corresponding Ack is received. If no Ack has been received for a given period, the complete packet is resent. Optionally, Nacks[21] may be sent back to the packet source to improve NICs efficiency. Furthermore, Nacks may be subdivided in UNacks[22] requesting to cease reemissions of the packet, and RNacks[23] triggering the immediate reemission of the packet.

In a fault-tolerant context, the NIC also needs to provide a fault-tolerant protocol that supports retransmissions and packet losses. For instance, in [53], a complete fault-tolerant NIC and its fault-tolerant message passing protocol are described.

### 2.1.5 Metrics

A brief explanation of the metrics utilized in this chapter is given in this section. While this provides sufficient information for the chapter's reading, the interested reader may find more in-depth details about these metrics and their analysis in Section 2.6.

**Performance**

In order to assess the performance of a NoC, there exist 3 widespread metrics inherited from the networking community.

The **injection rate** is the amount of packets that the simulator tries to push into the interconnect over a given period. Since this metric is a simulation parameter, it is generally represented on the abscissa axis.

The **throughput** corresponds to the effective amount of packets transmitted by the interconnect over a given period.

The latency roughly corresponds to the delay incurred by packets. Depending on the NIC protocol, it may have two different definitions. The **one-way latency** is the delay from the injection of the packet into the interconnect until the packet is received at the target node. For NICs that utilize acknowledgments, the **two-way latency** is the delay from the storage of packet at the source NIC until a corresponding Ack is received at the source node. In this chapter, the later will be utilized, since our fault-tolerant interconnect requires Acks.

Unfortunately, in the presence of faults, the latency and throughput metrics fail to support completely the analysis of interconnect's performance. In effect, they do not display the impact of lost packets. In this chapter, we utilize a node efficiency metric to quantify the impact of losses. This metric shows

[20] positive Acknowledgement
[21] Negative Acknowledgements
[22] Unreachable Negative Acknowledgements
[23] Retransmit Negative Acknowledgements

how many clock cycles were spent by nodes for sending packets that were successfully transmitted and how many cycles were wasted for sending packets that were eventually lost.

#### Fault tolerance

In this chapter, the fault tolerance of routing algorithms will be measured as the percentage of packet instances lost specifically because of the routing algorithm's limitations.

This metric provides a fine-grain vision of the routing capacity to overcome defect patterns, though ETE schemes may reduce the fault tolerance observed by the system through retransmission. During the simulation, all packets' instances sent to the interconnect are classified into 4 categories, following discussion in Section 4.4.3:

1. Transmitted: When the instance was received by its target node.
2. Intrinsic loss: When the instance was lost directly because of the interconnect defect pattern.
3. Partial loss: When the instance was lost partially because of the routing algorithm.
4. Routing loss: When the instance was lost directly because of the routing algorithm.

## 2.2 State of art

A state-of-art of routing algorithms is proposed in the following section. Fault oblivious algorithms are discussed in Section 2.2.1, and the most common deadlock freedom and adaptivity approaches for the 2D-Mesh topology are depicted. Then, fault-tolerant routing algorithms are surveyed in Section 2.2.2.

### 2.2.1 Fault-oblivious routing

#### Topology and communication awareness

The impact of routing function on the NoC[24] performance partly depends of its complexity and its degree of awareness. Ideally, an omniscient routing function might anticipate and arrange packets in an optimal way. In practice, global knowledge offers high performance at the expense of either high silicon overhead or intimate knowledge of the application [54, 27, 55]. On the opposite, topology-agnostic routing algorithms surveyed in [56] have limited information about the interconnect topology itself, which grants extreme versatility to these algorithms. Unfortunately, efficiency is generally lower than topology-aware routing algorithms, in particular if the topology at hand has some unexploited level of regularity. As a consequence, 2D-Mesh interconnects mostly use specific routing algorithms, even though most of them may be extended to 3D-Meshes or 2D-Torus.

#### Turn restrictions

In the context of meshes, the deadlock freedom of routing algorithms is a major concern. As a consequence, the choice of the deadlock freedom strategy by itself essentially defines the routing function. The Odd-Even model was proposed in [57], based on the turn model [37], and uses different complementary turn restrictions, depending on the location of the router to cumulate deadlock freedom and

[24] Network on Chip

adaptivity. Routers in even columns forbid East $\rightarrow$ North and North $\rightarrow$ West turns, while routers in odd columns forbid West $\rightarrow$ North and North $\rightarrow$ East turns. This approach achieves the same level of adaptivity, but since restrictions are distributed more evenly, higher performance is achieved in practice. In [58], segment routing is proposed. In that scheme, turn restrictions are set differently on each router. Compared to other topology-agnostic routing algorithms, this scheme takes advantage of the interconnect redundancy when possible, which improves significantly its performance. The turn restrictions are computed off-line, since they require complex computations, during which the interconnect is divided into segments (respectively regions).

#### Congestion awareness

The local knowledge (i.e. only neighbor nodes) provides a scalable interconnect but poses challenges regarding routing adaptivity. Hence, most adaptive algorithms are based on the the knowledge of 2-hops neighbors in order to make more sensible decisions, of course at the expense of some extra wires [59]. In [60, 61], congestion-aware routing algorithms are proposed, based on the Odd-Even scheme. The routing decision is based not only on the direction of the target node, but also on the comparative level of fillness of neighbor routers. In [62], the Neighbors on Route strategy is devised. This technique utilize the knowledge of the 2-hops away neighbor nodes to improve performance. In [63], the Regional Congestion Awareness is proposed to improve even more the efficiency of congestion-aware decisions. In this scheme, congestion information is aggregated from router to router in dedicated registers. In [64], the DyAD paradigm is proposed. In this scheme, each router supports both a low-complexity deterministic routing and a congestion-aware adaptive routing algorithm. This combination offers the low latency of deterministic routing algorithms for low traffic, and the efficiency of congestion-aware routing under high traffic. In [65], routing offers natural and graduated congestion awareness, based on an ant colony algorithm. Some training (ant) packets are propagated in the interconnect to exchange routers knowledge.

### 2.2.2 Fault-tolerant routing

There is a large variety of fault-tolerant routing strategies, due to the stringent complexity constraints on one side, and the large background from the computers network community on the other side. A partial classification of fault-tolerant routing algorithm is proposed below, yet many algorithms actually exploit multiple concepts simultaneously.

#### Turn restrictions customization

By far, the most common fault-tolerant approach is to adapt turn restrictions to the context of faults, either by augmenting the available adaptivity or by removing some restrictions around faulty nodes. In [66], Fick *et al.* propose to selectively remove some turn restrictions to improve the fault tolerance of the interconnect under faults. In effect, turn restrictions around faulty regions may be removed without jeopardizing the NoC's deadlock freedom. This is an elegant solution to improve the fault tolerance of the low-complexity turn restriction scheme, but it incurs a global configuration phase when a new fault occurs and deadlock occasionally arise. [67] proposes a similar approach that does not require complete reconfiguration but provides only results for one-fault configurations. In [68], the interconnect is split logically in 2 VNs[25] with different turn restrictions (Odd-Even and Inverted Odd-Even), and each packet is duplicated at source NIC[26] before one copy is sent in each part. In [69],

[25] Virtual Networks
[26] Network Interface Circuit

an adaptive deadlock-free routing algorithm, called NARA, is presented. Two VNs (North-Last and South-Last) are introduced to cumulate high adaptivity and deadlock freedom.

#### Look-ahead signals

In general, the presence of faults creates patterns that require specific actions from the routing algorithm. Typically, some packets will reach a configuration that requires routing adaptivity to bypass faulty routers and links. As a consequence, the designer may have to provide techniques for packets to bypass these faulty resources and reach their destination. Alternatively, additional information (such as look-ahead signals) may be given to the routing algorithm, so that appropriate decisions are taken in advance, and "difficult" configurations are reached less often. The latter approach improves substantially the average fault tolerance of the system, but do not cure the interconnect from pathologic cases, especially when the size of interconnect and the rate of defects increase.

Routing algorithms based on look-ahead signals consequently provide a larger fault tolerance with comparable routing mechanisms. For example, the widespread LBDR (Logic-Based Distributed Routing) [59] utilizes signals from the 2-hops away routers do avoid "difficult" configurations. [70] also proposes to utilize 2-hops away information to improve the performance under defects, by avoiding unnecessary hops.

#### Fault block model

Another widespread approach is to group faulty resources into regular **fault blocks**. A low-complexity routing is used safely outside these blocks. Healthy routers belonging to a fault block are either deactivated or reached through custom routing rules. In [71], a fault-tolerant routing based on the fault block model is proposed for $n$D-Meshes. This algorithm offers high adaptivity based on 2 VNs. The fault block model is refined to limit the number of deactivated routers in high dimensional meshes. Yet, [72] proposes a combination of XY routing for fault-free regions and rule-based turn restriction around faulty routers.

#### *f-strings* and *f-chains*

The so-called *f-string* and *f-chains* approach offers medium fault tolerance by surrounding faulty resources by detour routes. In that scheme, packets that would normally traverse faulty routers are misrouted around the faulty routers, following a chain (or a string) of routers. This scheme usually requires that some fault-free routers are deactivated in order to create manageable fault patterns, alike the fault block model. In addition, this approach suffers from congestion in the *f-chains* when traffic grows up. In this context, several improvements were proposed to allow routing towards deactivated routers, based on more complex routing and inter-router information exchange [73, 74, 75].

#### Explicit routing

In [47], a fault-tolerant explicit routing algorithm is proposed. Packets are routed by following a route defined at the source node. When new faults occur, affected routes are detected and updated accordingly. This approach alone suffers from route discovery costs, since alternative routes are found at random, but a combination with previously presented algorithms may improve the interconnect performance, as discussed in Section 2.5.1.

### Stochastic

A stochastic approach was also applied to fault tolerant routing under versatile shapes. In [76, 77, 78, 79], packets are routed following the gossiping approach. Each router transmits multiple copies of in-transit packets, in order to obtain a satisfying probability that packet reaches the destination. Several improvements are proposed, but this scheme requires large amounts of unnecessary copies of each packet, which translates into energy and maximum traffic degradation, as shown in [80].

### Deflection

Since deflection routing [21, 22, 23] does not suffer from deadlock hazards, there is little limitation to its adaptivity, and therefore its fault tolerance potential. In [81], deflection routing is used for routing packets in the presence of transient and permanent faults, however this approach suffers from two shortcomings in the presence of faults. First, the average number of hops increases faster than conventional routing algorithms. Hence, the chances to traverse a faulty resource is larger. Second, when available routes require multiple non-minimal routing decisions, the transmission of packets is only stochastic, and may actually fail under large fault rates.

### High fault tolerance

Most fault-tolerant routing algorithms are rarely able to guarantee packets transmission above fault rates of 10 20%. In effect, this property, which we will call high fault tolerance, requires a highly complex routing algorithm. To our knowledge, the only existing highly fault tolerant algorithm is presented in [82]. The proposed routing algorithm, named uLBDR, is based on packet forking and VCT[27] scheme. The authors claim that uLBDR provides *full coverage*, but either livelocks might appear or some fault patterns may not be covered.

### Routing tables update

While routing tables are generally considered overly inefficient for regular NoCs, it worth noting that several contributions from the computer network community offer high fault tolerance based on them. In [83], a reconfiguration mechanism for multicomputer systems is presented. This solution requires 2 routing tables, 2 VCs[28] and a global reconfiguration upon the occurence of a fault. In [84], NetRec, an algorithm for the dynamic reconfiguration of multiprocessing systems is given. The main advantage of NetRec is that the dynamic reconfiguration involves only a few routers around the faults, thus reducing the impact of the reconfiguration.

### Extension to 3D-Meshes

As the 3D-Mesh topology gains momentum, many 2D-Mesh fault-tolerant routing algorithms have been extended for supporting 3D-Meshes as well. A first approach is to handle vertical connections specifically and utilize an existing 2D-Mesh routing for planar routing [85, 86]. Alternatively, turn model may be adapted to support such topologies [87], but special care is required for ensuring the deadlock freedom of the interconnect.

[27] Virtual Cut Through
[28] Virtual Channels

## 2.3 Context

### 2.3.1 Motivations

We consider a many-cores chip, based on a 2D-Mesh NoC[29] as shown in Figure 2.8. Following the introduction from Section 2.1, the potential of such devices largely depends on designers' ability to provide a low-latency interconnect with high throughput capabilities. Moreover, future manufacturing processes will soon demand prohibitive validation and large guardbanding to preserve chips from the effects of variability. In that context, interconnects would rely on overly conservative designs, unless some level of fault tolerance is inserted.

In effect, introducing fault-tolerant interconnects allows for relaxing validation constraints and narrowing guardbands, since the interconnect is able to cope with sporadic faults without jeopardizing the system functionality.

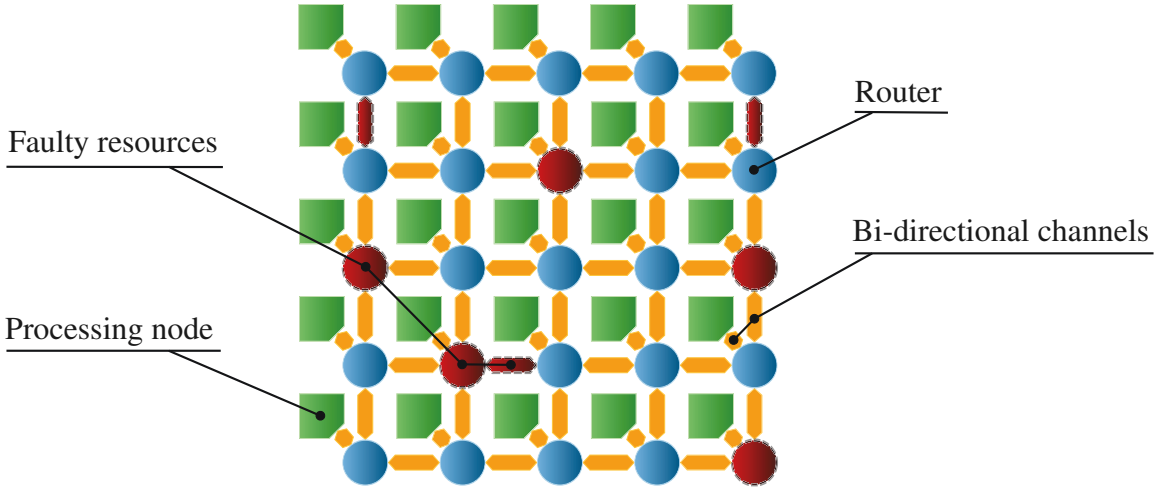



Figure 2.8: An example of faulty 2D-Mesh NoC which requires fault-tolerant routing

This chapter presents a set of routing algorithms targetting interconnects such as Figure 2.8. Firstly, Our contribution addresses permanent faults occuring dynamically in routers and transient faults hitting the links, through a dynamic recovery technique presented in Section 2.4.1. Second, Section 2.4 presents 3 routing algorithms that offer different levels of fault tolerance to many-core chips. Third, Section 2.5 displays improvements for limiting the degradation of the interconnect performance in the presence of faults. An extensive set of experimentation is then presented in Section 2.6. Finally, several formal properties of these techniques are given in Section 2.7.

### 2.3.2 Proposed router architecture

In this chapter, all proposed algorithms rely on the router architecture presented in Figure 2.9. The router essentially consists of 6 ports, a crossbar, and a control logic. 4 external ports are connected to neighbor routers through a bi-directional link. The local and VS[30] ports are respectively connected to the node's NIC[31] and the router crossbar (more details are given in Section 2.4.3). Each router's ports are shared between several VCs[32], as discussed in Section 2.1.3. The crossbar is a point-to-point connection to all router's ports. On top of these elements, the control logic manages routing

[29] Network on Chip
[30] Virtual Source
[31] Network Interface Circuit
[32] Virtual Channels

and arbitration operations. Finally, the router embeds a Link control module that handles transient and permanent link faults. The Link check circuit is responsible for the link state diagnosis, as detailed in the next section. When necessary, the Port deactivation circuit inhibits all traffic in the port.

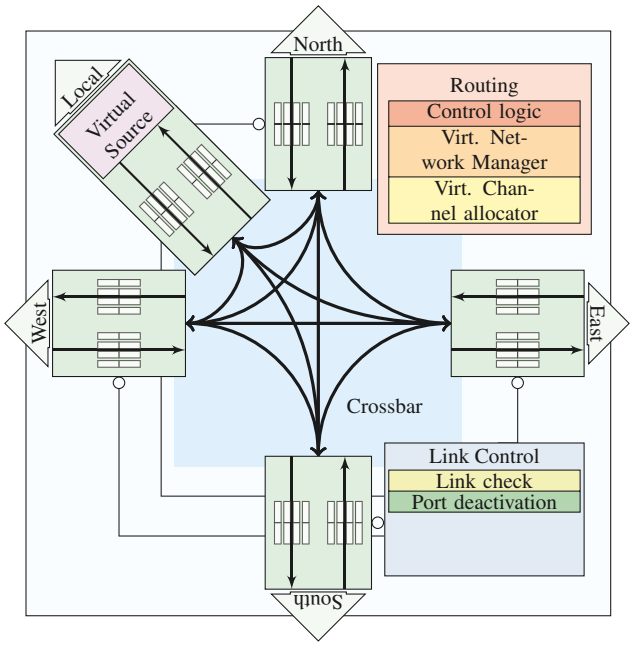



Figure 2.9: Router architecture details

### 2.3.3 Formalism

In order to present our contributions, we need a few definitions. From a formal perspective, a targeted processor chip $\mathcal{C}$ consists of a matrix of $W \times W$ computation nodes. In effect, following the 2D-Mesh topology of the interconnect, each interconnect node is associated with a couple $(i, j)$. Each node $\mathrm{n}$ itself is associated to a router ($\mathrm{Rt}$), a NIC ($\mathrm{Ni}$) and a vector of cores ($\mathrm{Co}$), as discussed in Section 2.1. Each router itself consists of a crossbar and a vector of ports ($\mathrm{P}$). Each port consists of a vector of channels $\mathrm{Ch}$, which contain in-transit flits.

Since the structure of objects defined above is quite complex, a hierarchical association operator $\triangleright$ is defined in Definition 2.3.1, for retrieving objects associated to a 'parent' one.

**Definition 2.3.1** (Association operator)**.** *Given an object $X$, the association operator $\triangleright$ is defined such that $X \triangleright \mathsf{Y}$ returns the unique object associated to $X$, refered-to by the key $\mathsf{Y}$. This operation reads as 'the object $\mathsf{Y}$ of $X$'.*

Based on the previous definitions, the figure gives the structure of processor-related objects considered in this chapter.

For the sake of readability, directions are defined formally in Definition 2.3.2. For instance, the South port of a router $\mathrm{r}$ may be accessed through $\mathrm{r} \triangleright \mathsf{P}[\mathsf{South}]$ or $\mathrm{r} \triangleright \mathsf{P}\left[\vec{S}\right]$.

**Definition 2.3.2** (Directions)**.** *Within a 2D-Mesh interconnect, relationship between nodes and their associated router follow usual cardinal points. As such, each router's port is identified by its target direction. While directions are necessarily integers for complying with vector access syntax, they will be denoted indifferently by full directions names ($\mathsf{South}, \mathsf{East}, \mathsf{North}, \mathsf{West}$) or their vector equivalent (respectively $\vec{S}, \vec{E}, \vec{N}, \vec{W}$). These external ports define the external directions set $\mathcal{D}_{ext}$. Internal ports (i.e. local port to NIC and VS defined in Section 2.4.3) are also associated respectively to direction ($\mathsf{Local}, \vec{L}$) and ($\mathsf{VS}, \vec{V}$) and form the set $\mathcal{D}_{int}$.*

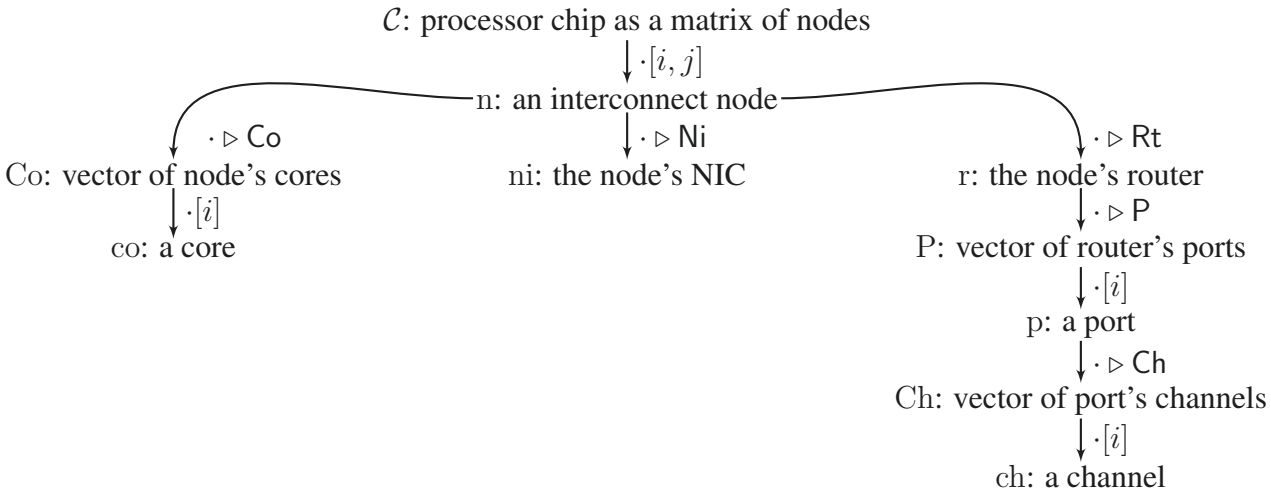



Figure 2.10: Formal structure of the considered processor, with objects name and relationship

In addition to directions, the availability of neighbor nodes is captured in Definition 2.3.3, 2.3.4 and 2.3.6. In effect, from a formal perspective, adaptive routing algorithms at least require the knowledge of nodes that are 1-hop away from current node.

In particular, routers embed a fault detection mechanism as described in Section 2.1.4, in order to avoid defective routers and links. The timely detection of faults is out of the scope of this thesis, thus perfect detection will be assumed in the following. In Definition 2.3.3, only output directions corresponding to a node with functional router and fault-free link are considered available.

**Definition 2.3.3** (Neighbour availability)**.** *We denote* $\mathrm{DirectionOK}\,(\mathrm{n}, d)$ *the boolean set to* true *iff the port of node* n*'s router corresponding to direction* $d$ *can be used for routing a packet. If* $d \in \mathcal{D}_{ext}$*,* $\mathrm{DirectionOK}\,(\mathrm{n}, d) = \mathsf{true}$ *means that there exist a neighbor node for node* n *in direction* $d$*, and its router and link with* n *are fault-free. Alternatively, if* $d = \mathsf{Local}$*,* $\mathrm{DirectionOK}\,(\mathrm{n}, d) = \mathsf{true}$ *only if the local port and the node* n *are fault-free. Finally,* $d = \mathsf{VS}$ *and* $\mathrm{DirectionOK}\,(\mathrm{n}, d) = \mathsf{true}$ *imply that the VS port is fault-free and has an available channel, following Definition 2.4.4.*

**Definition 2.3.4** (Neighbour)**.** *We denote* $\mathrm{Neighbor}\,(\mathrm{n}, d)$ *the neighbor node for node* n *in direction* $d \in \mathcal{D}_{ext}$*. This function is defined only when* $\mathrm{DirectionOK}\,(\mathrm{n}, d) = \mathsf{true}$*.*

**Definition 2.3.5** (Node Output directions)**.** *The set of output directions* $\mathrm{Outputs}\,(\mathrm{n})$ *contains external directions to nodes 1-hop away from the node* n*.*

Equivalently, output directions can be defined from neighbors existence through Equation 2.1.

$$\mathrm{Outputs}\,(\mathrm{n}) = \{d \in \mathcal{D}_{ext} | \mathrm{DirectionOK}\,(\mathrm{n}, d) = \mathsf{true}\} \tag{2.1}$$

**Definition 2.3.6** (Node Neighbourhood)**.** *The neighbourhood* $\mathrm{Neighbors}\,(\mathrm{n})$ *is defined as the set of nodes, which contains nodes 1-hop away from the node* n*.*

Equivalently, the node neighbourhood can be defined from neighbors through Equation 2.2.

$$\mathrm{Neighbors}\,(\mathrm{n}) = \{\mathrm{Neighbor}\,(\mathrm{n}, d)\,, d \in \mathrm{Outputs}\,(\mathrm{n})\} \tag{2.2}$$

Finally, packet routes are defined in Definition 2.3.7.

**Definition 2.3.7** (Route)**.** *A route from node* $s$ *to* $t$ *is a vector of directions* $(d_1, d_2, ..., d_k)$ *that represents a sequence of hops for a packet to reach target node* $t$ *from source node* $s$*.*

Under this form, routes only require 2 or 3 bits per hop, depending on the encoding technique. In addition, the set of routers already traversed by the packet may be computed from the route iteratively.

Posing $\mathrm{n}_i$ the $i$-th router visited by a route $R = (d_1, d_2, ..., d_k)$, Equations 2.3, 2.4 describe the route connectedness property, used for traversed routers computation.

$$\mathrm{n}_0 = s \tag{2.3}$$
$$\mathrm{n}_i = \mathrm{Neighbor}\left(\mathrm{n}_{i-1}, d_i\right), i \in [1, k] \tag{2.4}$$

Following the connectedness properties expressed in Equations 2.3, 2.4, the set of traversed routers is defined in Definition 2.3.8. This set is used by some proposed routing algorithms, in order to figure out whether routers were already traversed or not.

**Definition 2.3.8** (Traversed nodes set)**.** *For a given route $R = (d_1, d_2, ..., d_k)$, the node set* $\mathrm{Traversed}(R)$ *returns the set of nodes which router has been traversed by the route, including its source node.*

In addition to Equations 2.3, 2.4, a valid route requires Equations 2.5, 2.6 to hold, in order to effectively transfer a packet to its target node $t$.

$$\mathrm{DirectionOK}\left(\mathrm{n}_{i-1}, d_i\right) = \mathsf{true}, \forall i \in [1, k] \tag{2.5}$$
$$\mathrm{n}_k = \mathrm{t} \tag{2.6}$$

## 2.4 Gaining high fault tolerance

In this section, we present 3 fault-tolerant routing algorithms and a dynamic fault recovery mechanism. These algorithms share the same base ideas, but the addition of different fault tolerance techniques lead to significant behavioral changes. The Variant A presented in Section 2.4.2 is the simplest algorithm. In Section 2.4.3, the Variant B offers a higher fault tolerance, at the expense of increased complexity. Finally, the Variant C is proposed in Section 2.4.4. This algorithm is able to route packets in the presence of any set of multiple nodes and links faults, as long as a route exists. Compared to the existing solutions, the proposed algorithm provides fault tolerance without using any routing table. It is scalable and can be applied to multi-core chips with a 2D-Mesh core interconnect of any size. The algorithm is deadlock-free and avoids infinite looping in fault-free and faulty 2D-Meshes. Besides,the dynamic fault recovery technique presented in Section 2.4.1 guarantees that the interconnect recovers from the apparition of permanent and transient faults.

### 2.4.1 Dynamic recovery from permanent and transient faults

In this thesis, we consider that interconnects are subject to the frequent apparition of faults, as discussed in Section 2.1.4. Transient faults are usually tolerated through the use of ECC[33] in the end nodes combined with packets re-emission when necessary. In this section, we propose an holistic approach to handle the dynamic faults, based on Nack[34] emission.

The fault detection is out of the scope of this work, and is generally based on flits coding and BIST[35] at the router level. Nevertheless, whatever fault detection technique is used, packets may be split as a consequence of faults. Therefore, the interconnect should ensure that split packets do not endanger

[33] Error Correcting Code
[34] Negative Acknowledgement
[35] Built-In Self-Test

the interconnect behavior. Firstly, no flit shall remain in the router buffers infinitely, even after the occurrence of a fault. Second, with wormhole routing, the packet's head flit locks buffers as it goes, and these locks hold until the corresponding tail flit is received. Thus, if no action was taken, the occurence of faults may compromise the interconnect behavior by leaving some buffers locked infinitely. In addition, the packet source node has to re-emit the packet, either after timeout or after reception of a Nack.

The proposed technique consists of four steps. First, when payload and tail flits are received while no output VC[36] is locked - which stem from an illegal flit sequence - , the flit is drained. Second, when a transient or permanent fault occurs, the fault is detected by neighbor routers either through a fault detection signal, or an ECC attached to each received flit as presented in Section 2.1.4. Third, neighbor router(s) insert a specific so-called **Orphan tail flit** in input buffers associated with the faulty link or router. Fourth, in the case of a permanent router or link fault, the corresponding output buffers and outgoing VC buffers' locks are flushed before an hypothetical recovery.

The obtained behavior is presented in Figure 2.11, and shows no limitation regarding the occurrence of faults. In effect, packets are modified each time they are affected by a new fault. Afterward, the head slice of the packet flows naturally to the destination node and Orphan Tail flit(s) ensure that VC buffers are unlocked. The tail slice is drained since VC buffers have been unlocked by Orphan tail flits or output port flush. Finally, destination NIC[37]'s routine checks (e.g. tail flit type, packet size, ECC) detect the packet corruption, and return a Nack to the source node. However, if a link becomes permanently faulty after the packet head flit entered the upstream output port, the head flit is drained, and no Nack is returned. In this rare case, the source node will re-emit the packet only after timeout occured.

The proposed technique handles naturally special cases, where a Local or a VS port is faulty. In effect, if source node's Local port is hit by a transient or a permanent fault, while emitting a packet, the source node's router will drain flits in the Local port and insert an Orphan tail flit in the selected output port, enabling a part of the packet to reach the destination. Consequently, the destination node will detect the corruption of the packet, and return a Nack. If the Local port fault is permanent, the Nack will be drained by the source node's router. Alternatively, when the destination node's Local port becomes faulty, the router simply drains packets targeting this port, and the source node will re-emit the packet after timeout. When utilized VS[38] buffers are hit by a fault, the scheme is similar. Due to the VS buffer function, either packets are partly in the VS buffer and previously used ports, either the end of the packet remains in the VS buffer and forthcoming ports contain the header and following payload flits. Following the dynamic fault recovery technique, the router containing the faulty VS buffer will drain packet flits that did not enter the VS buffer yet and insert an Orphan tail flit on the selected ouput port, if any. If the VS buffer fault is permanent, it is deactivated and avoided by later routing decisions.

In Figure 2.11, a packet is sent from a source node to a target node (*i*). On the left side (respectively right side), a transient link fault (respectively permanent router fault) occurs in (*ii*), splitting the packet in 2 parts. In (*iii*), an Orphan tail flit is inserted after the defective link. The tail part of the packet then advances and is drained before the defective resource in (*iv*). Finally, the head packet part reaches the target node, which detects that a fault occurred based on the Orphan tail flit signature. Ultimately, a Nack is returned to the source node, initiating a packet retransmission.

[36] Virtual Channel
[37] Network Interface Circuit
[38] Virtual Source

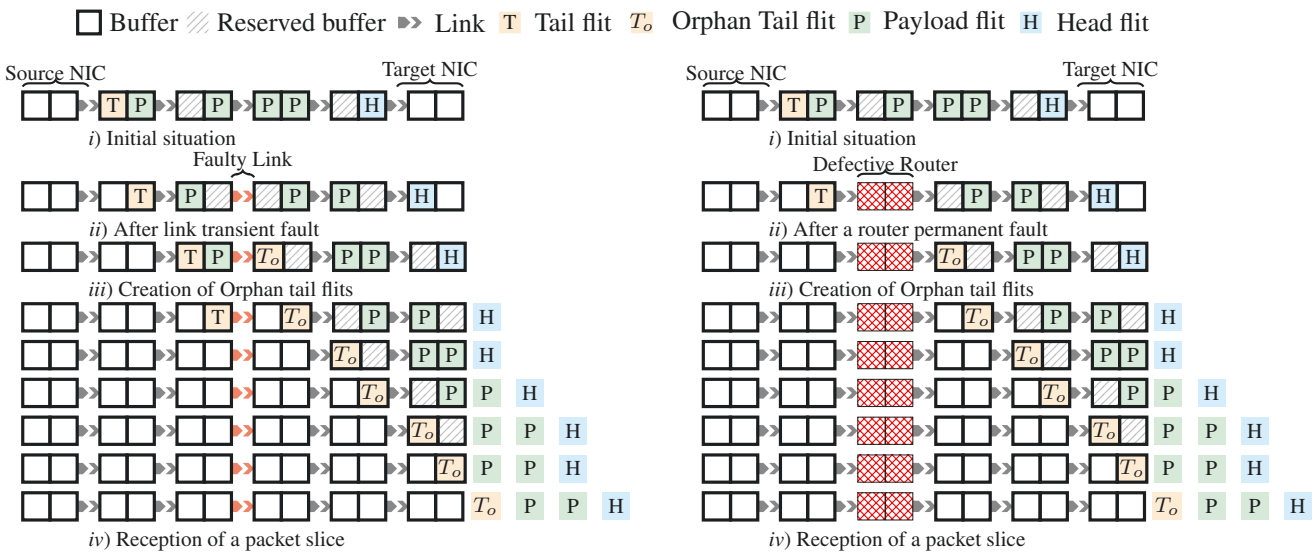



Figure 2.11: Proposed dynamic recovery technique under transient link and permanent node faults

### 2.4.2 Variant A: Low-cost fault tolerance

The Variant A is the simplest routing algorithm proposed in this thesis. The obtained fault tolerance is reasonable and overpasses many state-of-art fault-tolerant algorithms. Its low complexity may be leveraged for low fault-rate situations and/or low reliability targets.

This routing algorithm essentially relies on two mechanisms. First, 2 different VNs[39] are used to propagate packets without deadlocks, as presented in Section 2.4.2. Second, the routing is based on a hierarchy of output ports, depending on the packet destination, as shown in Section 2.4.2. Based on these elements, the algorithm is presented in Section 2.4.2 and several experimental results are given.

#### Virtual Networks

The proposed routing algorithms rely on the dichotomy and isolation of interconnect resources. To this end, VCs are grouped to form VNs. VNs are defined as non overlapping groups of VCs, where packets are propagated. In this thesis, we used 4 VCs per port, 2 being assigned to each VN, but more VCs may be allocated to each VN in order to improve the NoC[40] performance. The VNs to use for each packet is chosen during its injection in the interconnect, depending on the position of the target node, in order to reduce the impact of the VN's turn restrictions.

#### VN turn restrictions

In the proposed algorithms, deadlock freedom is obtained through restrictions on the routes. In effect, avoiding certain turns (i.e. sequences of ports' direction in a packet route) effectively prevents the formation of cyclic buffer dependencies. In this work, the deadlock freedom of algorithms is proven through the use of link numbering, but the implementation is based on turn restrictions. Moreover, the specificity of these algorithms is the use of two different (and complementary) turn restrictions schemes, and therefore two different link numberings. This combination improves the adaptivity of the algorithms, while limiting the complexity overhead.

[39] Virtual Networks
[40] Network on Chip

In order to avoid deadlocks, two specific link numberings are used, based on works given in [69]. The proposed routing algorithms combine 2 VNs with each a different numbering named respectively North-Last and South-Last. These link numberings do not allow packets going north (respectively south) to turn another time following Definition 2.4.1, hence effectively preventing the apparition of deadlocks.

**Definition 2.4.1** (Link numbering)**.** *A number $a$ is attached for each link of the interconnect, according to Figure 2.12. During its routing, a packet stores the numbering $a_l$ of the last used link. Afterwards, the routing algorithm must always choose a link with a larger number $a_n$.*

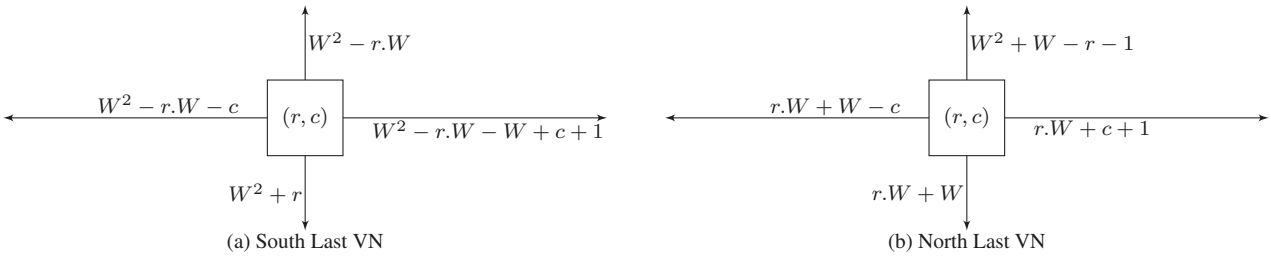



Figure 2.12: Link numbering for a 2D-Mesh interconnection network of dimension $W \times W$

Numbering formulas given in Figure 2.12 are generic regarding the size $W \times W$ of the mesh. They may also be extended to rectangular meshes, but since square meshes are far more common, the simpler version for square meshes was presented. A link numbering example is presented in Figure 2.13 for a $4 \times 4$ mesh.

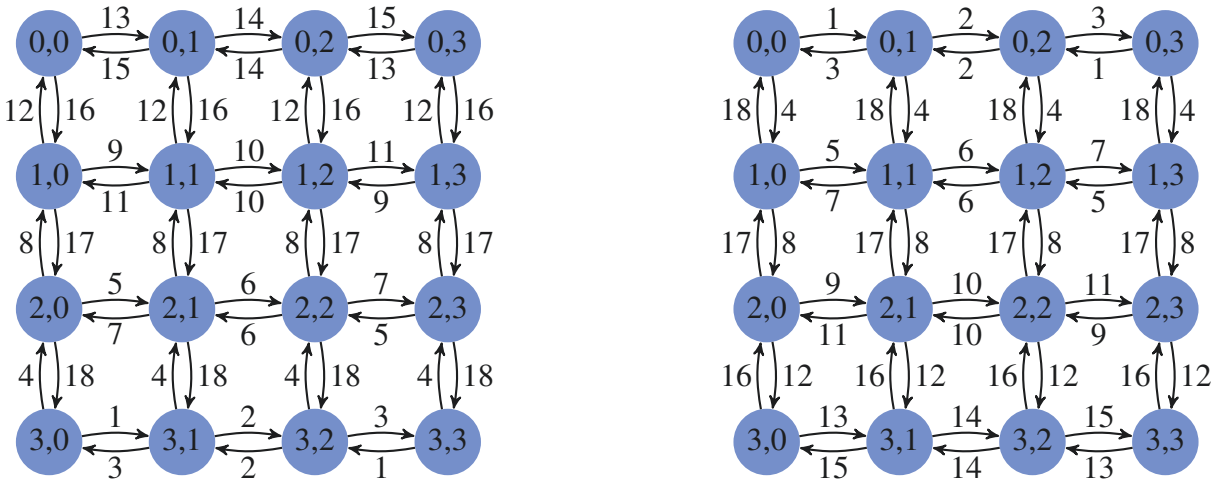
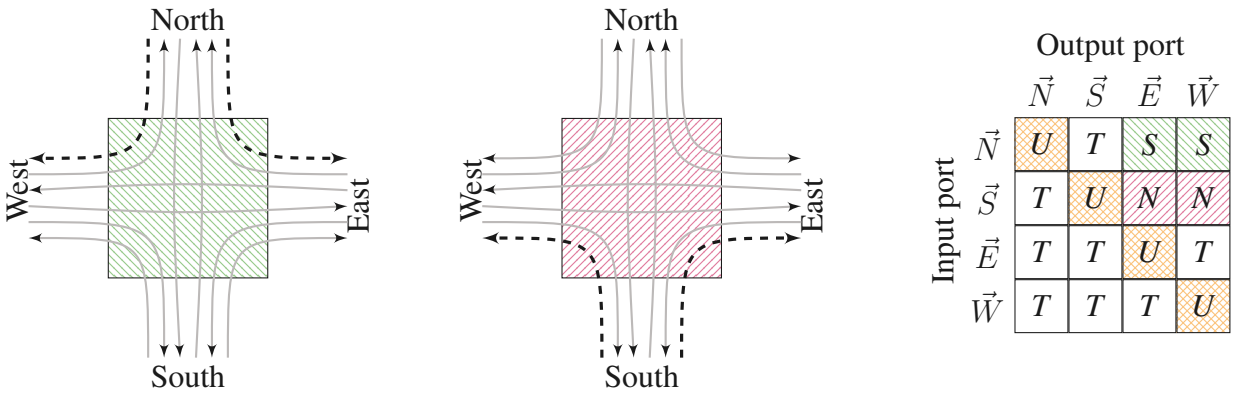



Figure 2.13: South-Last and North-Last link numbering for $4 \times 4$ 2D-Meshes

While link numberings provide a solid theoretical background for the deadlock freedom of the proposed algorithms, this scheme is relatively expensive to implement in practice. In effect, link numbers must be compared at each routing step and the last number has to be embedded in the packet's head flit. Therefore, turn restrictions are used for implementation.

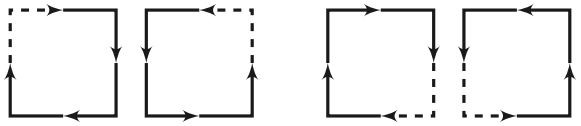
This approach is equivalent to the link numbering regarding the prohibited turns as shown in Figure 2.14-a,b, but is much more economical. In effect, it takes only a $4 \times 4$ table to check if the Virtual Network forbids a given turn, as shown on Figure 2.14-c. Turns marked *U* are forbidden for both VNs (U-Turns are prohibited). North-Last VN restrictions are marked *N* and South-Last VN restrictions are marked *N*. Other turns, including those which source or destination is Local, are granted (marked *T*).




(a) South-Last VN (b) North-Last VN (c) Turn restriction table

Figure 2.14: Turns restrictions for South-Last and North-Last VNs

In Figure 2.15, we show turns that are prohibited with this approach. We can see that each cycle is broken as a consequence of a turn prohibition. The VN to use is chosen each time a packet is injected in the interconnect. If the destination is northern (respectively southern) of the source, South-Last (respectively North-Last) VN will be chosen. If the source and destination nodes are on the same row, VN is chosen depending on VC use.



(a) North-last VN (b) South-last VN

Figure 2.15: Effect of prohibited turns on VNs dependency cycles

Compared to the corresponding link numberings, the turn restrictions are more restrictive since they disallow all U-turns. In effect, each link numbering allows some u-turns. For instance, following Figure 2.13-a, a packet may flow from router $(0,0)$ to $(0,1)$ and back. This does not jeopardize the interconnect deadlock freedom, because once the packet has returned to $(0,0)$, the link numbering forbids to use again the same link. From a practical perspective, this is generally not a desired behaviour though, since the packet wastes energy and links bandwidth. As a consequence, turn restrictions do have the same deadlock freedom guarantees, and disable some unwanted turns.

Based on the turn restrictions discussed above, the allowed node neighbourhood is defined in Definition 2.4.2. This is the set of output directions considered for routing packets, as shown in Algorithm 1.
**Definition 2.4.2** (Allowed output directions)**.** *We denote* $\mathrm{Allowed}_p(\mathrm{n})$ *as the subset of* $\mathrm{Outputs}(\mathrm{n})$ *that contains only the directions where the packet* $p$ *can be forwarded, based on the turn restrictions of the VN used by* $p$*.*

### Output hierarchy

The routing algorithm described in Algorithm 1 is based on the Output Hierarchy given in Definition 2.4.3. $Q$ is defined as the set of output directions that a packet may use with respect to the VN turn restrictions and excluding the last visited direction $d_{in}$. The packet will be forwarded to the output direction $d_{out}$ after the routing decision.

**Definition 2.4.3** (Output Hierarchy)**.** *Given $Q$ the set of eligible output ports of the current node $n$, and $t$ the packet target node,* $\mathsf{Hierarchy}(Q, n, t)$ *returns the highest direction of $Q$ in the hierarchy presented in Figure 2.16.*

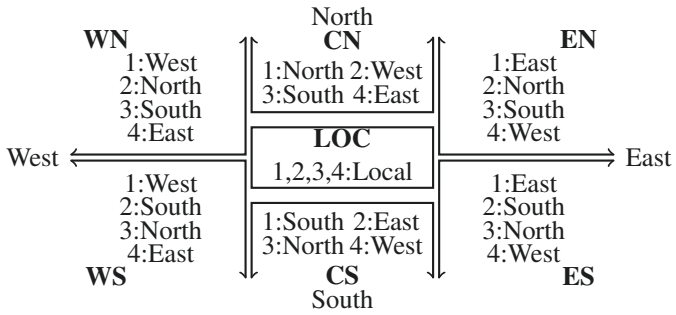



Figure 2.16: Output hierarchy

The Output Hierarchy given in Definition 2.16 and Figure 2.16 leverages the interconnect regularity to route packets toward their destination in the presence of multiple faulty routers and links. Depending on the destination node position relative to the current node, output ports are requested in an appropriate order. When destination is on the same column, South (CS), North (CN) or Local (LOC) output port is promoted.

The Output Hierarchy alone is not sufficient to guarantee 0% packet loss, even for low defect rates. In effect, because the routers have only a partial knowledge of the interconnect state, packets are sometimes trapped, due to faults in the interconnect. In such circumstance, more complex routing decisions are required, as discussed in next sections.

### Presentation

In Algorithm 1, the Variant A routing is presented in an algorithmic form. At line 6, the Ouput Hierarchy selects an output direction amongst those allowed by the packet's VN.

**Algorithm 1** Variant A algorithm for a packet $p$ from the source node $s$ to the target $t$ at node $n$

**Require:** $n$: current node for packet $p$
**Require:** $d_{in}$: direction of $p$'s input port
**Require:** $Q$: set of eligible output port direction
**Require:** $d_{out}$: output direction to route $p$ to
1: **procedure** ROUTINGVARIANTA
2: Set $Q$ to $\text{Allowed}_p(n)\}$
3: **if** $n = t$ **then**
4: Target is reached! Route to Local port.
5: **else if** $Q \neq \varnothing$ **then**
6: Set $d_{out}$ to $Hierarchy(Q, n, t)$
7: Route packet to direction $d_{out}$.
8: **end if**
9: Stop because target is unreachable!
10: **end procedure**

With the Output hierarchy, output directions are ranked depending on the relation between the position of the packet target node $t$ and the current node $\mathrm{n}$. The routing algorithm will select the highest direction in the hierarchy, which is usable to forward a packet. Starting from the highest (top) direction, the router scans each direction in the *strict descending order*. For each direction, the router checks that the corresponding router exists and is neither faulty nor forbidden by the VN. For example, if destination is East North of the current router, the hierarchy is $(\mathsf{East}, \mathsf{North}, \mathsf{South}, \mathsf{West})$. This means that the router will successively prefer the East, North, South and West outputs, until it finds a direction to forward the packet (i.e. neither faulty nor incoming direction). If all conditions are met, the direction is chosen, and the packet can be routed.

In some cases, even the lowest (bottom) direction is not eligible for forwarding the packet. That is, all options have been exhausted without obtaining an usable direction. In a such condition, the normal routing is stopped, in order to avoid infinite looping. In next sections, more advanced routing algorithms are designed to cope with this situation.

The Variant A routing algorithm requires several features from the NIC. At injection time first, NIC has to select the packet's VN. If the target is northern (respectively southern) of the source, South-Last (respectively North-Last) VN will be chosen. This scheme minimizes the probability to eventually drain the packet. For example, when target is northern, there are little chances that packet had to be routed to a South port. Hence, the VN's turn restrictions are unlikely to disturb the transmission of the packet. Second, this algorithm requires the emission of Acks[41] upon reception. Otherwise, emitter NIC shall resend the packet upon timeout or Nack reception.

[41] positive Acknowledgements

### Results

The Variant A algorithm leads to a small router logic, but when defects occur in the interconnect, packet losses increase, and the interconnect performance decreases dramatically. In effect, the throughput observed in Figure 2.17 falls from $0.065$ to $0.032$ for 5% of faults and $0.0032$ for 20% of faults.

This drastic diminution of the achievable throughput can be explained by the very low efficiency of the routing algorithm under defects. In effect, up to 14% of the packets sent to the interconnect are lost due to routing limitations, according to Figure 2.18. Since NICs have to wait for the timeout associated to a lost packet, before sending another instance or another packet, the efficiency of the interconnect reduces rapidly, as discussed in Section 4.4.1.

Nevertheless, the routing algorithm is suitable for low fault rate (e.g. 5%), where it transmits most of the packets successfully. In addition, the average latency observed with this algorithm is more stable than for more complex ones.

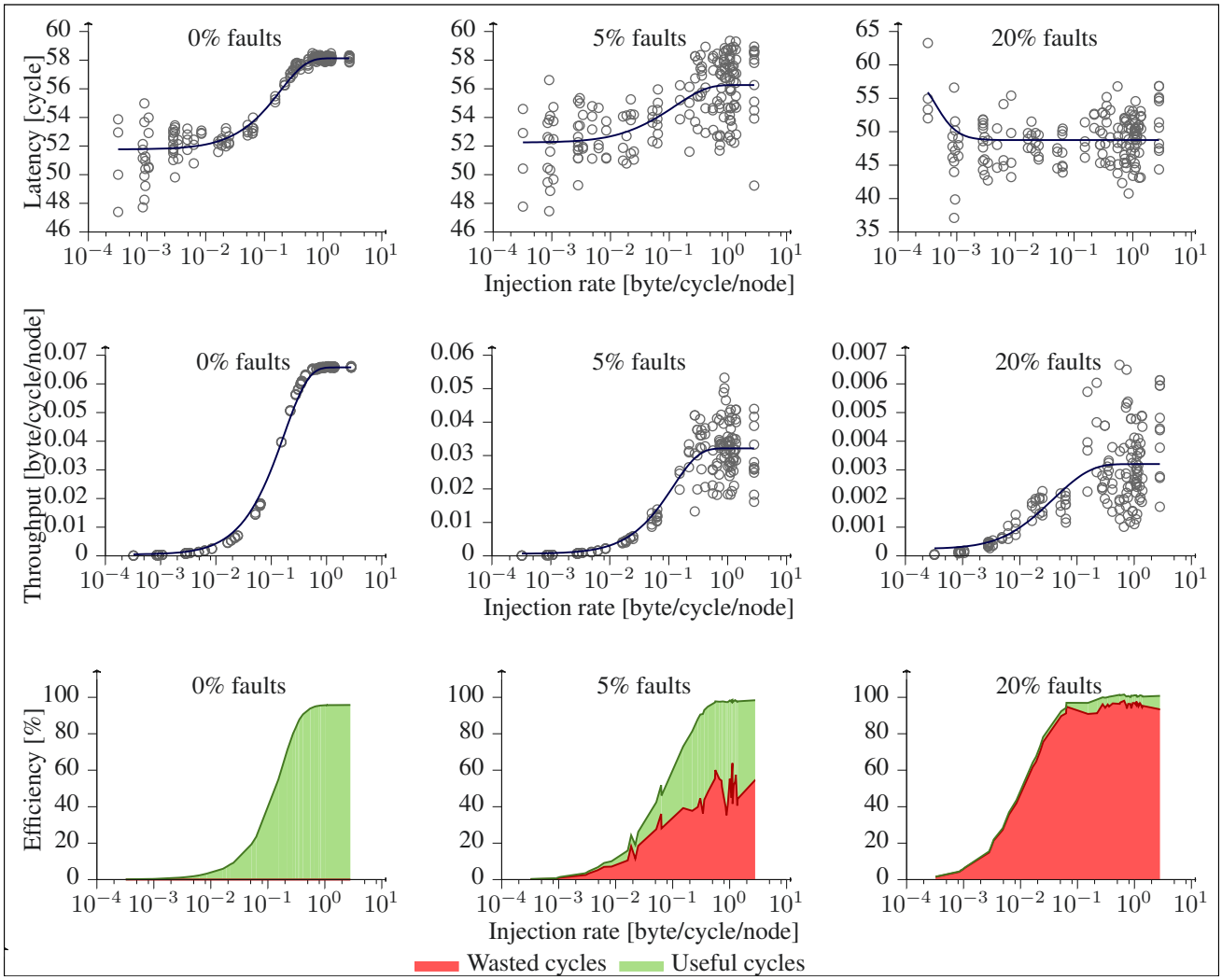



Figure 2.17: Performance of the Variant A routing algorithm

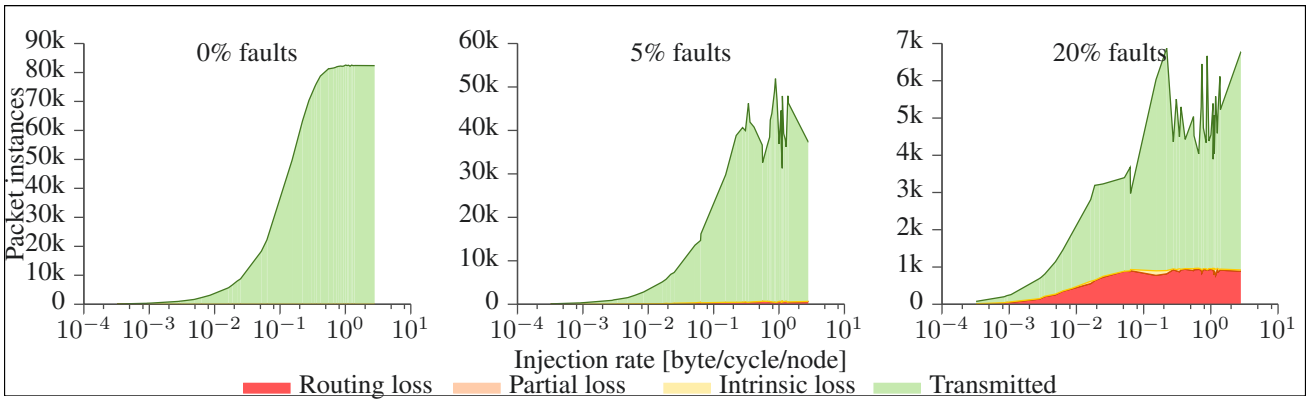



Figure 2.18: Fault tolerance of the Variant A routing algorithm

### 2.4.3 Variant B: More fault tolerance through virtual buffers

The Variant B routing algorithm is based on the Variant A algorithm presented in Section 2.4.2, with significant additions towards fault tolerance. In effect, the VS[42] buffers -discussed in Section 2.4.3- are inserted in routers, to cope with more fault patterns. Since this gain in adaptability may threaten the interconnect functionality, RS[43] is subsequently added to avoid packet livelocks.

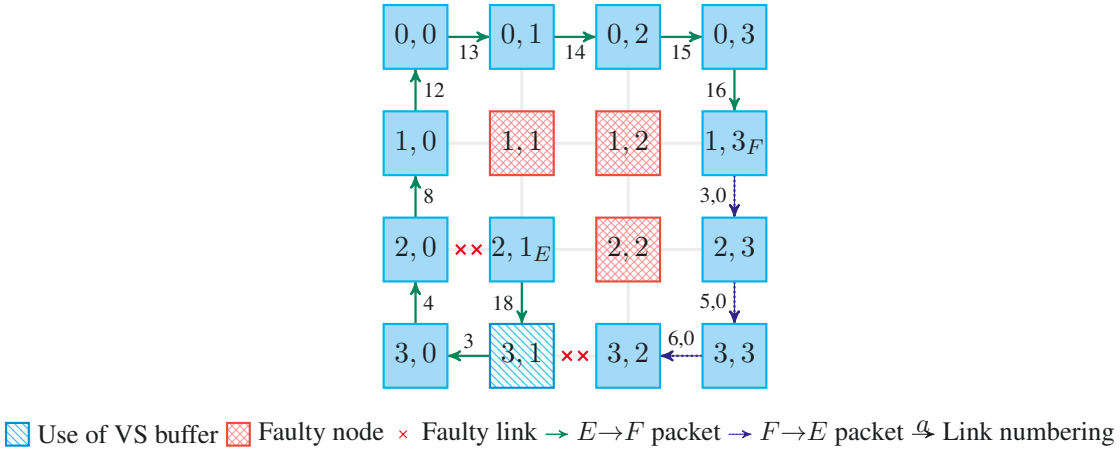



Figure 2.19: An example of Variant B algorithm utilization

In Figure 2.19, a packet from the node $E$ to the node $F$ has to take turns that are forbidden in the South-Last VN[44]. The packet transmission requires a VS on node $(3,1)$ to route the packet from node $E$ to node $F$, but the routing algorithm fails to route $F$ to $E$ packet. In the example, because target node is northern, South-Last VN is chosen first. However, a defect on node $(2,2)$ prevents the packet from reaching directly the target. At node $(3,1)$, this packet must take the South $\rightarrow$ West turn, which is prohibited in the South-Last VN. The entire packet is stored in the Virtual Source buffer and re-emitted later. The packet finally reaches the target node $F$ using the South-Last VN.

[42] Virtual Source
[43] Router Stamping
[44] Virtual Network

### Virtual Source

For reducing the number of lost packets, The support of VSs is added to Variant A, in order to exchange packets between VNs, based on [88]. Indeed, when there are faulty nodes and links, routing packets between 2 nodes may take many turns. When a packet cannot be routed without breaking its Virtual Network turn restrictions, the Virtual Source enables the packet to continue towards its target, as defined in Definition 2.4.4.

**Definition 2.4.4** (Virtual Source)**.** *The Virtual Source is used to swap safely packets from a VN to another, or simply to break dependencies within the same VN. The complete packet is stored in the VS buffer shown in Figure 2.9, thus eliminating the dependencies on previously visited buffers. Then, it is re-emitted starting from this node.*

The VS behavior intrinsically prevents deadlocks occurrence by isolating former and new VNs. As a consequence, the routing from/to a VS buffer is not subject to turn restrictions. Equivalently, when a packet traverses a VS, its current link numbering is reset, allowing any output direction for further routing decision.

From a theoretical point of view however, VS buffers have a limitation under high traffic. In effect, all VS channels may be used already when another packet requests routing through Virtual Source. Unfortunately, blocking the packet until a VS channel is available may result in a deadlock. The only reliable alternative is then to drain the new-coming packet, optionally after waiting for limited amount of time. Consequently, packets may be lost because of the routing algorithm and not directly faults. On the up side, this situation does not occur under fault-free interconnects, since routing only requires VS as a consequence of certain heavy fault patterns. From a practical perspective, packet draining due to VS channels exhaustion is very rare. Following Figure 2.20,2.21, the utilization of VS is required 18 times (respectively 90) for 100 packets routed using the Variant B algorithm (respectively Variant C). As a consequence, the probability that several packets do request VS at the same router simultaneously is very limited compared to the probability of an intrinsic loss. According to Figure 2.21, only Variant C looses 0.2% of the packets due to VS exhaustion in the presence of faults and for injection rates beyound saturation.

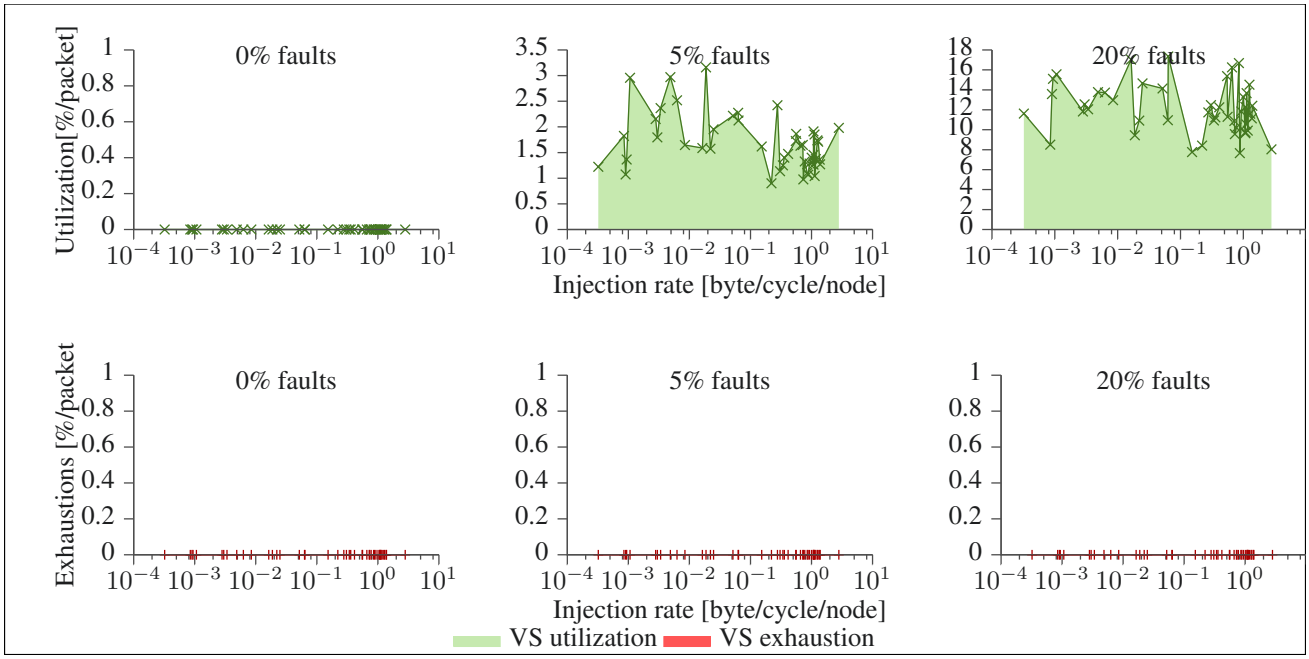



Figure 2.20: Normalized VS utilization and exhaustion for Variant B

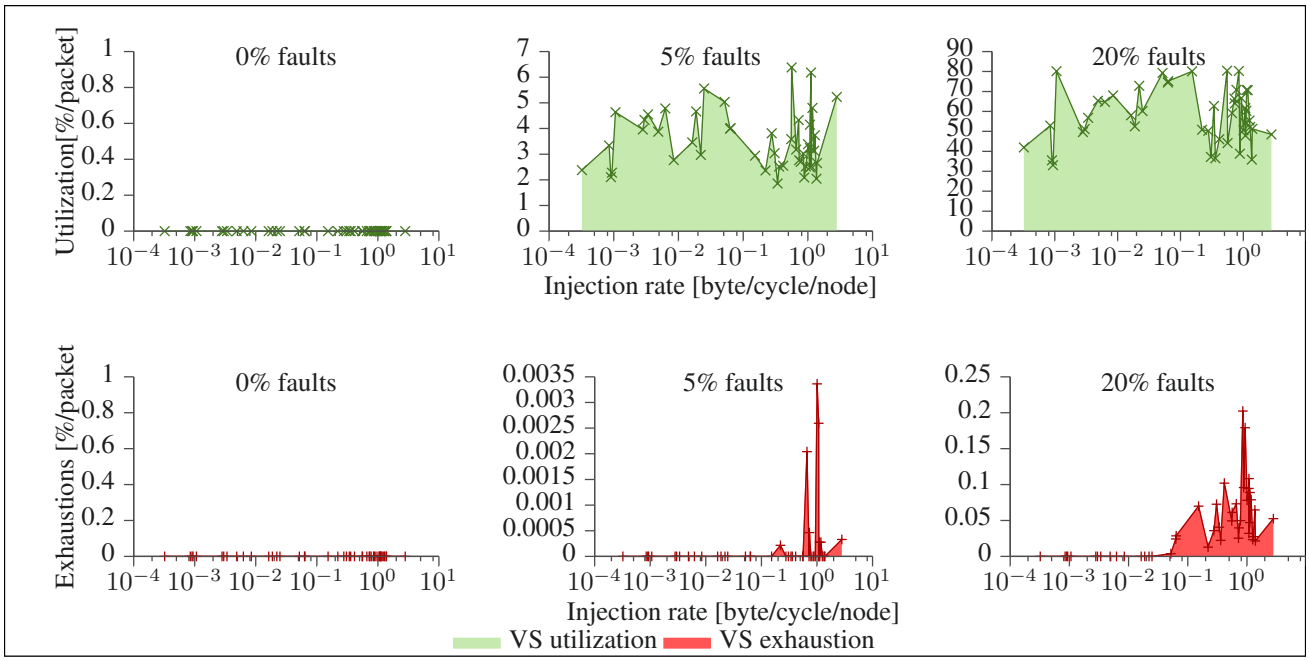



Figure 2.21: Normalized VS utilization and exhaustion for Variant C

### Router stamping

Unfortunately, adding VS requires an additional complexity for avoiding infinite looping. Therefore, RS is used, as presented in Definition 2.4.5.

**Definition 2.4.5** (Router Stamping)**.** *Each direction used by a packet $p$ during routing is stored with it (i.e. stamped), within the route $R_p = \{d_1, d_2, ..., d_k\}$ following Definition 2.3.7. When computing a new hop, the router ensures that the node corresponding to the output port is not already in the traversed node set* $\mathrm{Traversed}(R_p)$*, as defined in Definition 2.3.8.*

Practically, the computation of the set of eligible nodes $Q$ is based on the search of the neighbour nodes among traversed nodes set $\mathrm{Traversed}(R_p)$. This approach requires iterative comparison of traversed nodes with neighbors, but only 2 bits are added to the packet for each hop it takes.

Therefore, we build $Q$ by scanning iteratively the visited nodes. In fact, $Q$ is initialized to $\mathrm{Neighbors}(n)$. Then, the visited nodes are inferred iteratively and compared with the $Q$ set nodes. If there is a matching node, it is removed from $Q$.

### Presentation

Variant B algorithm is given in Figure 2. This is an enhancement of the Variant A algorithm defined in Algorithm 1, which includes the VS and RS. In effect, a VS is used when necessary, and the route is memorized by the packet in $P_m$. Consequently, packet cycles are avoided. Moreover, once the packet has reached its target node, the used route is available at the target node $t$. This will allow the creation of explicit routes in Section 2.5.

**Algorithm 2** Variant B algorithm for a packet $p$ from the source node $s$ to the target $t$ at node $n$

**Require:** $n$: current node for packet $p$
**Require:** $Q$: set of eligible output directions
**Require:** $d_{out}$: output direction to route $p$ to
**Require:** $R_p$: route used by $p$
1: **procedure** ROUTINGVARIANTB
2: Set $Q$ to $\mathrm{Neighbors(n)} \setminus \mathrm{Traversed}(R_p)$
3: **if** $n = t$ **then**
4: Destination is reached! Route to Local port
5: **else if** $Q \neq \varnothing$ **then**
6: Set $d_{out}$ to $Hierarchy(Q, n, t)$
7: **if** $d_{out} \in \mathrm{Allowed}_p(x)$ **then**
8: Add $d_{out}$ to $R_m$
9: Route packet to $d_{out}$ port
10: **end if**
11: **if** $\mathrm{DirectionOK\ (n, VS)}$ **then**
12: Route to VS port
13: **end if**
14: **end if**
15: Stop because destination is unreachable!
16: **end procedure**

### Results

In Figure 2.22, it can be observed that this additional overhead has a significant impact on the interconnect's performance, as the throughput under 20% of faults doubles. Indeed, the number of packets lost is reduced compared to Figure 2.18. Nevertheless, it can be noted that the average latency is growing compared to Variant A, as some packets drained by Variant A are now routed, and often require more time.

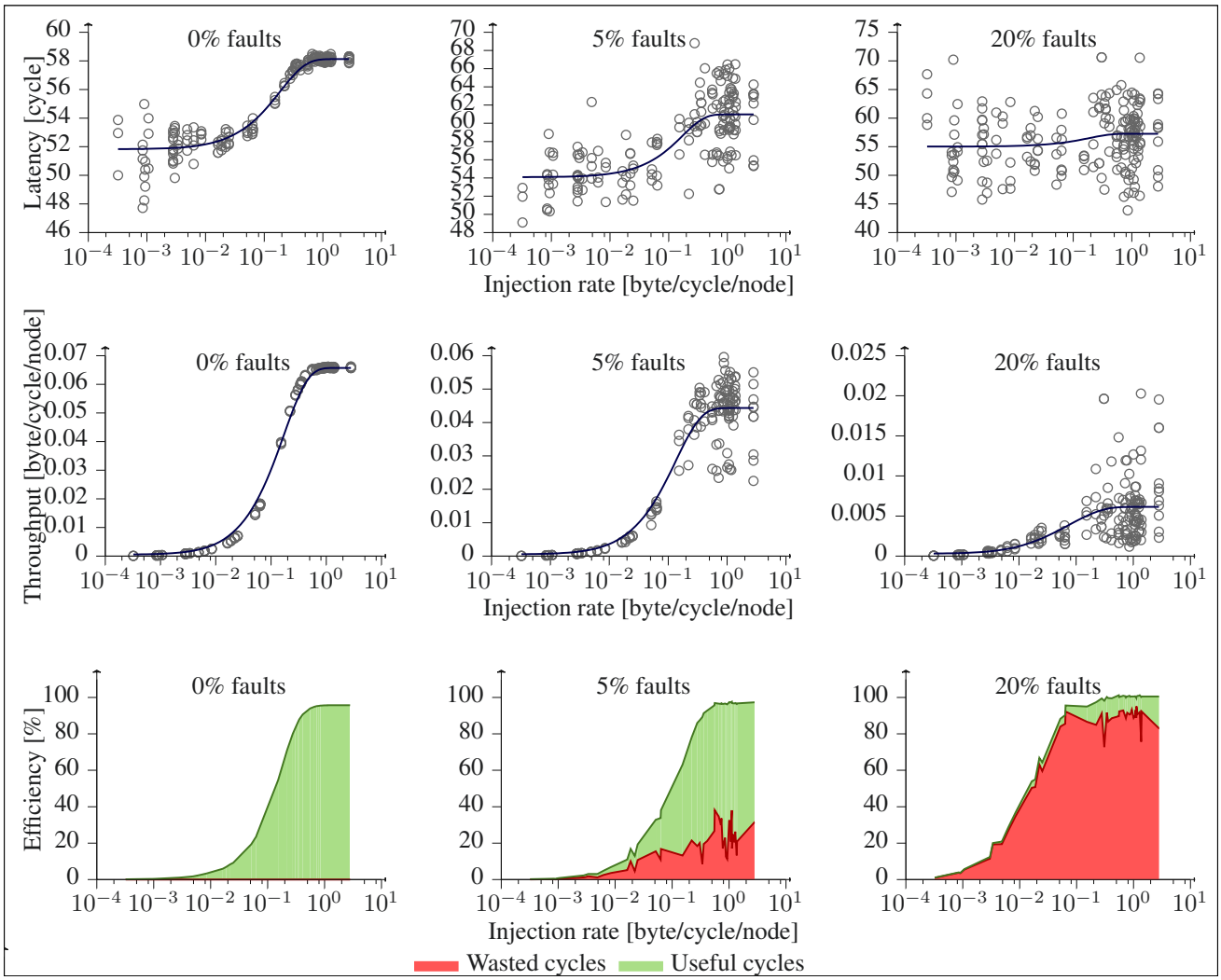



Figure 2.22: Performance of the Variant B routing algorithm

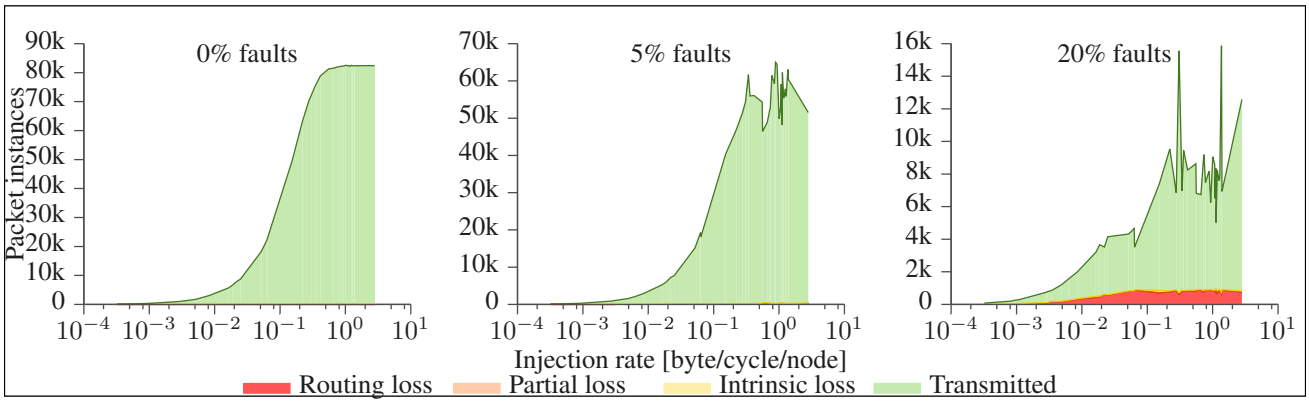



Figure 2.23: Fault tolerance of the Variant B routing algorithm

### 2.4.4 Variant C: Echo mode and high fault tolerance

The previous routing algorithm (Variant B) could potentially reach any destination as long as a route exists. Still, 0% packet loss could not be achieved. In effect, sometimes packets are trapped with the Variant B algorithm, because routers have only a partial knowledge of the interconnect. With Variant C, a VS[45] is used to break the buffer dependencies, and the packet is routed in Echo mode. This algorithm allows us to cope with the most demanding conditions, without using a routing table. In particular, it addresses cases where the interconnect is almost partitioned.

#### Echo mode

Echo mode virtually allows to route all packets as long as at least one route exists, under the limitation of the VS channnel exhaustion case described in Section 2.4.3. In essence, Echo mode rewinds the packet's route until an alternative route is found following Definition 2.4.6, and based on [89].

**Definition 2.4.6** (Echo mode)**.** *If there is no usable direction to forward the packet to the destination node using hierarchy defined in Definition 2.4.3, i.e. $Q = \varnothing$, then the Echo mode will be applied. The Echo Mode will enable the packet to rewind until it finds another route.*

During Echo mode, the direction corresponding to rewinded hops are removed from the current route $R_p$ and the "rewinded" router is stored in the echo nodes set $\mathrm{Echo}_p$. In ( 2.7), the relationship between $R_p$ and $\mathrm{Echo}_p$ is given, where $V_p$ is the set of all nodes visited by $p$ during its routing.

$$V_p = \mathrm{Traversed}(R_p) \cup \mathrm{Echo}_p \tag{2.7}$$

#### Presentation

The Variant C algorithm is presented at Algorithm 3, based on the Echo mode mechanism described above. This algorithm routes packets under Hierarchy mode mostly, but when necessary Echo mode is used to find a route to the target node. Under hierarchy mode, the packet's route $R_p$ is filled as the packet goes. Oppositely, Echo mode removes last hops from the route $R_p$ and memorizes nodes from which Echo mode was used. In addition, VS is requested when packet switches from Hierarchy to Echo mode, since packet makes a forbidden U-turn.

For example, consider the routing of the packet from the node $F$ to node $E$ in Figure 2.19. Based on the hierarchy, the packet will be routed to the South at the source node $F$. If Variant B was used, it would be trapped in the node $(3, 2)$. However, with Echo mode, the packet returns to the source node $F$, and try another direction. At this time, the set of echo nodes is $\mathrm{Echo}_p = \{(3, 2), (3, 3), (2, 3)\}$. First, the router starts by examining the hierarchy given in Figure 2.16. Since the destination node $(2, 1)$ is on the south west direction, the hierarchy is $(\mathrm{West}, \mathrm{South}, \mathrm{North}, \mathrm{East})$. West node is faulty and the South direction has already been used, so it is necessary to shift to the following direction, North. Since the router is fault-free, it can be used and the packet eventually reaches the target node $E$.

When the interconnect is partitioned, the Echo mode may not be possible, because the packet already to its source router. In this case, the algorithm stops at line 14 of Algorithm 3. For instance, consider the packet from the node $F$ to node $E$ in Figure 2.19, and assume that the node $(0, 1)$ is faulty. The packet returns to the source node $F$ after the *first Echo* and tries the North direction. Eventually, it returns to the source node leading to $\mathrm{Echo}_p = \{(3, 2), (3, 3), (2, 3), (0, 2), (0, 3)\}$. After this *second Echo*, the routing stops, because every possible options have been exhausted. In effect, in the hierarchy

[45] Virtual Source

**Algorithm 3** Variant C algorithm for a packet $p$ from the source node $s$ to the target $t$ at node $n$

**Require:** $n$: current node for packet $p$
**Require:** $Q$: set of eligible output directions
**Require:** $d_{in}$: direction of $p$'s input port
**Require:** $d_{out}$: output direction to route $p$ to
**Require:** $R_p$: route used by $p$
**Require:** $\mathrm{Echo}_p$: echo nodes set of $p$

```
1:  procedure RoutingVariantC
2:      Set V_p to Traversed(R_p) ∪ Echo_p
3:      Set Q to {d ∈ Outputs(n)|Neighbor(n, d) ∉ V_p}
4:      if n = t then
5:          Destination is reached! Route to Local port
6:      else if Q ≠ ∅ then /*Normal mode*/
7:          Set d_out to Hierarchy(Q, n, t)
8:          if d_out ∈ Allowed_p(n) then
9:              Append d_out to R_p
10:             Route packet to d_out port
11:         end if
12:     else/*Echo mode*/
13:         if n = s then
14:             Stop because network is partitioned!
15:         else
16:             if d_out ∈ Allowed_p(n) then
17:                 Pop d_out from R_p
18:                 Add n to Echo_p
19:                 Route packet to d_out port
20:             end if
21:         end if
22:     end if
23:     if DirectionOK (n, VS) then
24:         Route to VS port
25:     end if
26:     Stop because VS is unavailable!
27: end procedure
```

order, West is faulty, South and North have already been visited and are stored in $\mathrm{Echo}_p$, and East node does not exist. Consequently, the packet is drained, assuming that the destination is unreachable (i.e. *the network is partitioned*).

### Results

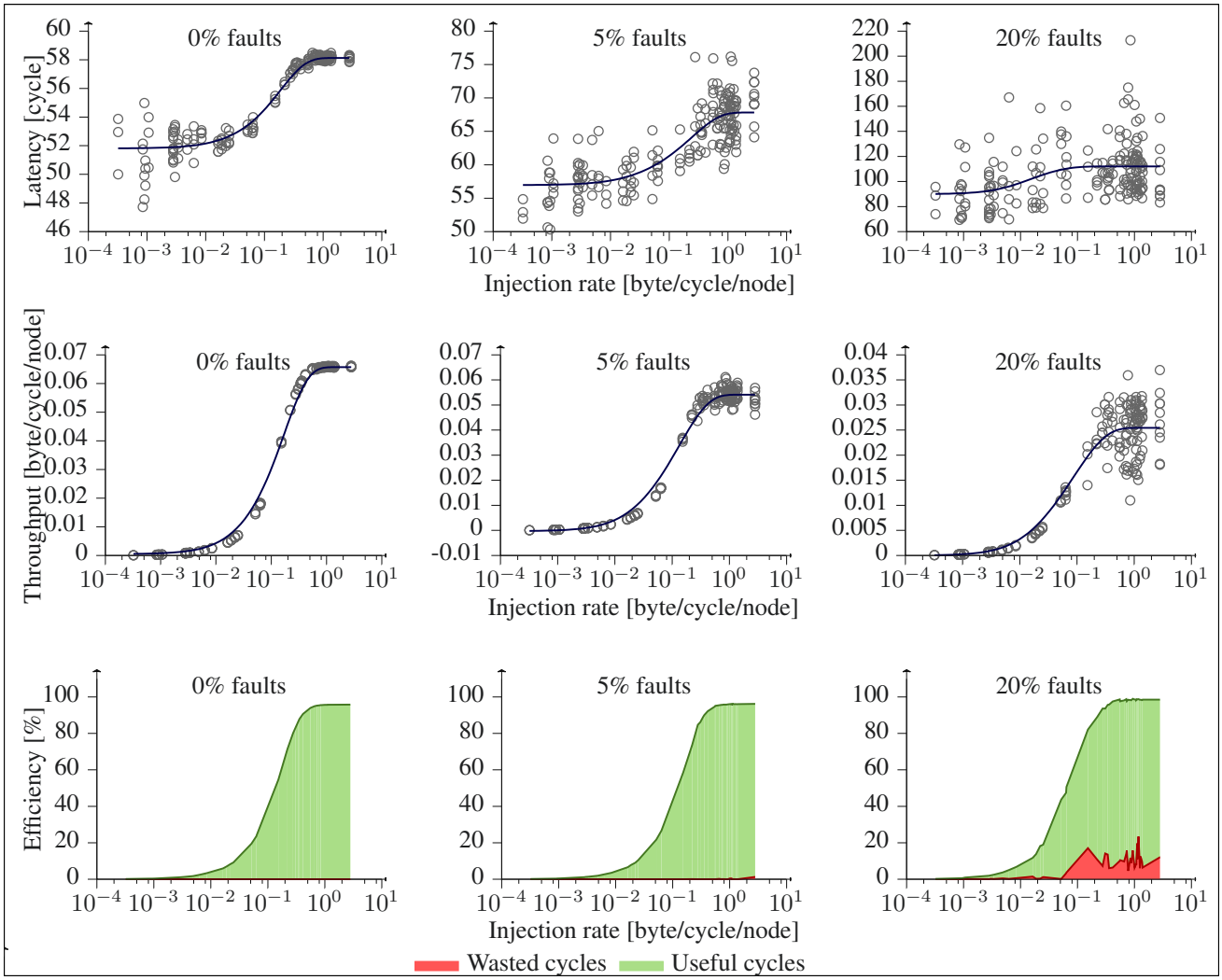



Figure 2.24: Performance of the Variant C routing algorithm

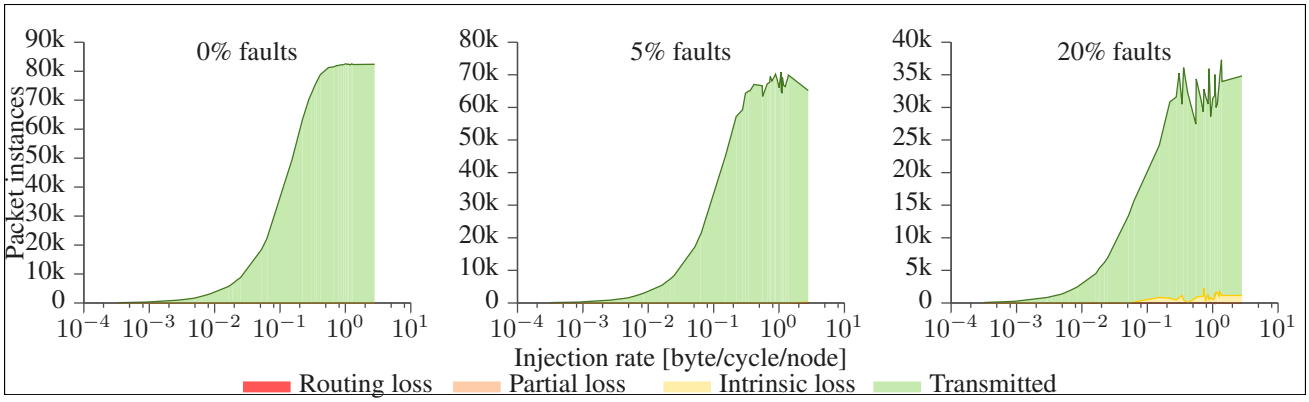



Figure 2.25: Fault tolerance of the Variant C routing algorithm

Variant C algorithm grants near-zero packet losses for each fault set, as shown in Figure 2.25. As a consequence, the efficiency of nodes increases and the throughput is closer to the ideal. In effect, Figure 2.24 shows that the saturation throughput is 0.065 for 0% faults, 0.054 for 5% faults and 0.025 for 20% of faults. In particular, the throughput obtained with Variant C is 10× higher than Variant A and 5× higher than Variant B.

## 2.5 Improving performance under faults

Future applications will require processors with many cores communicating through a regular interconnection network. Meanwhile, the Deep submicron technology foreshadows highly defective chips era. In this context, not only fault-tolerant designs become compulsory, but also their performance in the presence of defect becomes important.

In this section, several techniues are proposed to improve the efficiency of fault-tolerant routing algorithms presented in Section 2.4. First, the Explicit mode is proposed in Section 2.5.1 to improve performance when several packets are exchanged between the same source-target nodes. This is particularly interesting for streaming applications, which transfer huge amount of data between the same source and target nodes.

### 2.5.1 Addition of Explicit mode

The performance improvement is based on the fact that the route of packets previously routed with Variant C can be stored at source and target nodes. This is particularly beneficial when packets have activated Echo mode, as a consequence of defects. This will allow throughput to be significantly improved and thus improve the system efficiency. This is especially important for streaming applications that use repeatedly the same source- target node pairs with large amounts of data.

#### Explicit mode

In order to use Explicit mode, the **explicit route** is extracted from the RS[46] information after the first packet has been routed from a given source. The explicit route is then stored at the target node, according to Definition 2.5.1. The acknowledgment then follows the Explicit route, and reaches the source node with a copy of the explicit route. This allows the source node to store its own copy, for future packets. Because routes are stored in nodes' memory, there is virtually no limit to the storage of explicit routes.

**Definition 2.5.1** (Explicit route)**.** *For a given source node $s$, and considering a target node $t$, $ER_{s\to t}$ is a valid route from $s$ to $t$, obtained after a packet from node $t$ was received by the node $s$ or a packet from $s$ to $t$ was acknowledged.*

**Definition 2.5.2** (Explicit mode)**.** *When an Explicit route $ER_{s\to t}$ exists in node $s$, any packet $m$ from the node $s$ to the node $t$ will store the Explicit route $ER_m$, starting from the source node $s$. Afterwards routers will read directly the route $ER_m$, skipping Hierarchy and Echo Mode, according to the algorithm in Figure 4.*

#### Presentation

According to Algorithm 4, the proposed routing algorithm consists of 3 different modes. When possible, the router enables Explicit mode for packets and Acks[47] which improves the average latency. Otherwise, the *vanilla* Variant C presented in Algorithm 3 is utilized. Note that the Variant B algorithm may be used as well, but the input of Explicit mode is lower in that case.

For example, consider the routing of the packet from the node $S$ to node $T$ in Figure 2.26. Based on the hierarchy, the packet will be routed to the East straight to the node $(2, 3)$. Eventually, the packet

[46] Router Stamping
[47] positive Acknowledgements

**Algorithm 4** Variant C with explicit Mode for a packet $p$ from the source node $s$ to the destination $d$ at node $n$

**Require:** $d_{in}$: input direction of packet $p$
**Require:** $d_{out}$: output direction to route $p$ to
**Require:** $n$: current node for packet $p$
**Require:** $Q$: set of eligible output ports
**procedure** ROUTINGVARIANTCEXPLICIT
  **if** $ER_p \neq \varnothing$ **then** /*Explicit mode*/
    Pop the first element of $ER_p$ to $d_{out}$
    **if** $d_{out} \in \text{Allowed}_p(n)$ **then**
      Route packet to $d_{out}$ port
    **else**
      **if** DirectionOK (n, VS) **then**
        Route to VS port
      **end if**
      Stop because VS[48] is unavailable!
    **end if**
  **else**ROUTINGVARIANTC
  **end if**
**end procedure**

will be trapped in node $(3, 0)$ and start Echo mode. The packet returns to the node $(2, 1)$, and tries another direction. First, the router starts by examining the hierarchy, as given in Figure 2.16. Since the target node $(0, 3)$ is East North, the Output Hierarchy is $(\text{East}, \text{North}, \text{South}, \text{West})$. East has already been used, so it is necessary to shift to the following direction, North. Because the node is fault-free, it can be used to reach the destination node $T$. The proposed algorithm requires Echo Mode to route the packet from source node $S$ to target node $T$, but not for the acknowledgment, thanks to the Explicit mode.

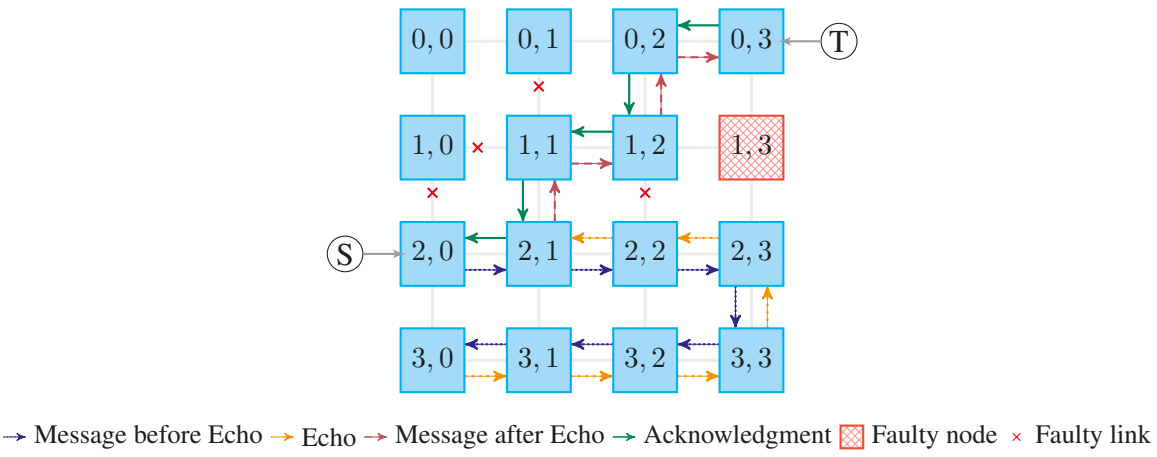



Figure 2.26: Motivational example for Explicit mode

If the interconnect were partitioned, the Echo mode would eventually return the packet to its source node's Local port, allowing the source node to detect the partitioning.

Afterwards, the acknowledgment of the packet from $S$ to $T$ is routed using Explicit mode, using the Explicit route from the acknowledged packet, as defined in Definition 2.5.2. Therefore, instead of choosing the East direction at node $(0, 2)$ based on the output hierarchy, the acknowledgement directly uses a shorter route and reaches the source node $S$. Any subsequent packet from the node $S$ to the node $T$ -and reverse- will follow directly this Explicit route too, improving substantially the traffic and average latency in case of faults.

**Results**

In order to show the interest of the Explicit Mode, we send bursts of 2 packets to the same target node successively from all nodes of the interconnect. As expected, there is little difference between the original Variant C and the version with Explicit mode for low fault rates, regarding the average route length (Figure 2.27-a) and traffic per packet (Figure 2.27-b). However, one can see that the proposed algorithm clearly improves the average route length and the generated traffic for fault rates starting from 20%.

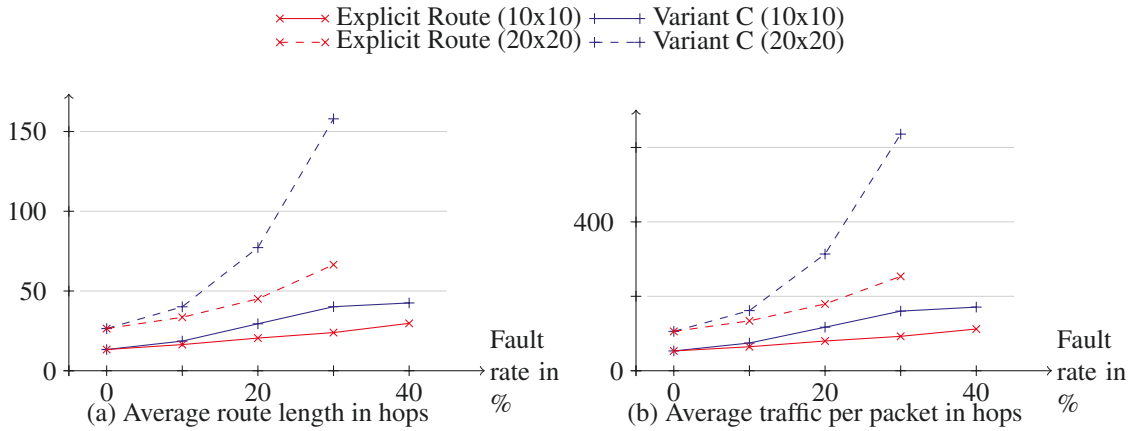



Figure 2.27: Comparison between original Variant C and version enhanced with Explicit mode

## 2.6 Experimental results

### 2.6.1 Simulation model

To support our work, a simulation model -detailled in Chapter 4- has been built with a focus on fault tolerance. This simulator has been created following the SystemC methodology, and is based on the router architecture presented in Figure 2.9.

Routers are simulated at the cycle level, which enables us to account for packets queuing and crossbar and link arbitration effects. We chose to use an uniform traffics and fault patterns. Results were obtained with packets of 4 payload flits. In our parametric model, we used 4 VCs[49] per port, 2 per VN[50] and input buffers of 4 flits per VC. Unless otherwise specified, no outstanding request was utilized. Based on these parameters, the injection rate at NIC[51] interface is one flit per clock cycle, i. e. 1[ns], when the previous packet's acknowledgment has been received.

The figures given in this chapter are obtained by averaging results from several fault patterns in each configuration. We considered *Node* faults sets where only the nodes can be faulty. Faults are induced during the warming period of the simulator and routers receive notifications when a neighbor node becomes defective. The retransmission of the packets is disabled in order to sensitize the interconnect behaviour to the quality of the routing algorithm.

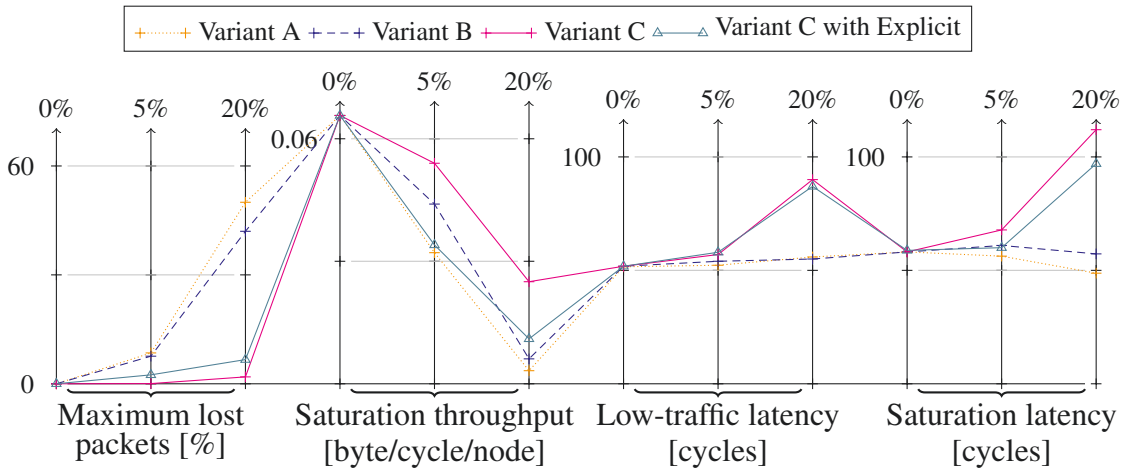



Figure 2.28: Global comparison of the proposed algorithms

### 2.6.2 Algorithm variants comparison

Figure 2.28 displays a multi-dimensional overview of the routing algorithms proposed in this chapter. Each group of 3 axes measures one characteristic of the routing algorithm, under different defect rates.

Regarding the rate of packets' instance loss, Variant A and Variant B algorithms provide a limited resilience, while the Variant C offers high fault tolerance. The fault tolerance of the Variant C with Explicit mode support is reduced since many explicit routes that were stored during warming are invalid due to later fault injection. In most cases however, a single retransmission would suffice to transmit successfully the packet.

The throughput observed at interconnect's saturation reflects the algorithms hierarchy for fault tolerance. In effect, our set-up shows that lost packets dramatically reduce the interconnect throughput by "blocking" NICs until they reach a corresponding timeout. Hence, our experiments tend to show that efficient retransmission schemes require a highly fault-tolerant routing algorithm.

The latency displayed in Figure 2.28 in turn mirrors the achieved throughput and fault tolerance. In effect, in the presence of faults, the routing of packets requires longer routes to bypass defective resources, and is hindered by the creation of bottlenecks in the interconnect. Hence, the *vanilla* Variant C and Variant C with Explicit mode both exhibit a high latency under defects, while the latency of Variant A and Variant B remains stable. The latency of routing algorithms with low fault tolerance may even decrease, since only packets with the closest source-target distance are transmitted, when the number of faults increases.

Overall, the results show that Variant A is fault-tolerant enough when fault rate remains low (e.g. 5%). Furthermore, when faults are more frequent or packets are critical, the fault tolerance of the interconnect could be improved by using Variant B. However, this algorithm may be insufficient, in particular for large meshes such as $10\times10$. In this eventuality, the Variant C guarantees near-zero lost packet for up to 20% of faults. The addition of Explicit Mode rather decreases the achieved performance in Figure 2.28 since the experimental set-up leads to many packet losses and few packets are sent with the same source-destination node pair. However, Figure 2.27 shows that in a different context, Explicit mode is able to improve substantially the interconnect's efficiency.

[49] Virtual Channels
[50] Virtual Network
[51] Network Interface Circuit

## 2.7 Routing algorithm properties

The routing algorithms Variant A,B and C, respectively presented in Section 2.4.2, 2.4.3, 2.4.4 have several important properties, which have been validated by means of simulation for different sizes of mesh and worst-case fault scenarios. First, they do not use any routing table. Second, the interconnect is resilient to dynamic and transient faults, thanks to the fault recovery mechanism presented in Section 2.4.1. Third, we claim that the proposed algorithms are safe. In effect, they are deadlock-free, do not generate cycling packets and always terminate.

Additionally, the Variant C algorithm leads to three possible outcomes. Most frequently, the destination is reached. Alternatively, a VS[52] channel exhaustion occurs following discussion in Section 2.4.3, and the packet is dropped. This end is not desirable and is rarely observed in practice. The last outcome is a drop after the packet returned to its source. A major property of the Variant C is that this type of packet drop only occurs when there is no valid route from the source to the target node. In brief, besides the rare VC[53] channel exhaustion, Variant C always find a route toward destination if there is one, or detects the partitioning and informs the source node.

In this section, we give several routing algorithm properties. First, we discuss the requirements A, B and C algorithms. Second, we claim that they are deadlock-free and do not generate packet cycles. Third, the termination of the different algorithms is given. Finally, the partitioning detection and the high fault tolerance of Variant C are discussed.

The described properties have been validated by means of simulation of different sizes of mesh and fault patterns. Moreover, we obtained a formal proof for each one. However, due to space constraints, it cannot be presented in this paper.

### 2.7.1 Memory requirements

It worth noting that Variant A, B and C algorithms do not require any routing table, according to definitions and algorithms presented in Sections 1, 2 and 3. This property stems from the fact that routing decisions follow the Output Hierarchy described in Section 2.4.2.

However, for Variant B and C, there is a requirement on the storage size of input ports. In effect, routing decisions require that the complete packet's route is stored in the router. Therefore, practical implementation of these algorithms would require to dimension the buffers of input ports in relation with the theoretical maximum route length (i.e. $W \times W$). Alternatively, routing may be limited to shorter routes, in order to reduce buffers size, but in that context, the fault tolerance would be degraded. In both cases, given $l$ the maximum supported route length, $W \times W$ the number of nodes in the processor, and $B$ the size of a flit in bits, the minimum depth of each input port channel for Variant B (respectively Variant C) is expressed in Equation 2.8 (respectively Equation 2.9).

$$\mathsf{Depth}_{VarB} \geqslant \frac{2\left(l + 2\log_2(W)\right)}{B} \tag{2.8}$$

$$\mathsf{Depth}_{VarC} \geqslant \frac{2(l+2)\log_2(W)}{B} \tag{2.9}$$

[52] Virtual Source
[53] Virtual Channel

### 2.7.2 Resiliency to dynamic and transient faults

For all routing algorithms proposed in this thesis, the resiliency to dynamic fault occurence is mainly supported through the mechanism proposed in Section 2.4.1. In effect, when a fault appears in the interconnect, the objective is to ensure that aftermaths will not degrade the interconnect function. If the newly defective resources are not used until fault is detected by neighbor routers, there are no consequences on the interconnect behaviour. However, if one or more packet(s) flits enter the newly defective area, involved routers will most probably enter an incorrect state and packets will be lost. In this section, we will show that the proposed fault recovery mechanism ensures that routers recover a correct state, and that lost packets are eventually re-emitted by their respective source nodes.

The first pitfall is router capacity loss. In effect, if a packet is transmitted while a fault occurs, ① the downstream router may expect a tail flit which will never arrive and ② packet flits may stall in the upstream router. As a consequence, channels allocated to this packet will remain locked by the router infinitely. Even worst, other routers will be impacted as well, alike deadlocks block progressively the interconnect. The solution to both issues relies on the fact that router drain flits which are not part of a legal sequence. In effect, wormhole routing demands that routers first receive a head flit, associate an output port channel to the input port channel, then receive payload flits and one tail flit in the end, to finally free the allocated port channel. Whenever a payload or a tail flit is received at one input port channel which is not associated to any output channel, flit cannot be routed anymore and is consequently drained.

Problem ① is solved by clearing input channels connected to the newly faulty resource and inserting a specific -Orphan- tail flit Subsequently, this artificial tail flit will follow the head slice of the packet and allocated channels in downstream routers will be freed on its way. Issue ② is settled by clearing output channels connected to the newly faulty resource and ongoing channel associations toward these channels. As a consequence, the tail slice will be drained and following packets will bypass this faulty resource thanks to the fault-tolerant routing.

The second pitfall is packet loss. This issue is solved at NIC[54] level, through packet re-emission triggered on timeout. In effect, the target node has to return an Ack[55] upon packet reception. Hence, if the source node does not receive any Ack after a given time period, assumption is made that packet was lost on its way, and packet is resent. For performance and stability reasons, this time period is generally large compared to the average packet latency. Therefore, an alternative scheme is also implemented for efficiency reasons. In the presence of transient and dynamic faults, packets extracted from the interconnect may be corrupted following bit flips or packet truncation. When target node receives a corrupted packet, it returns a Nack[56] to the source, which immediately re-sends the packet, reducing significatively the overall packet latency.

### 2.7.3 Safety

According to Sections 2.4.2, 2.4.3 and 2.4.4, the proposed algorithms are safe. In effect, variants A, B and C guarantee that, in any condition:

- No cyclic dependencies amongst router buffers exist, which would enable deadlocks otherwise.
- Packets cannot cycle infinitely in the interconnect.

[54]Network Interface Circuit
[55]positive Acknowledgement
[56]Negative Acknowledgement

The deadlock freedom is supported by two elements shared between the proposed routing algorithms. First, packets are transmitted inside one VN[57] of choice, which obeys to turn restrictions. The VN mechanism guarantees that no deadlock spans between packets from different VNs. Second, for Variant B and C, the transfer of packet between VNs is based on VSs which do not allow deadlocks.

In the Variant A, livelock freedom is supported by turn restrictions, and its associated link numbering. In the Variant B, packets RS[58] guarantees that packets enter only once in a given router. Finally, in Variant C, packets can enter a given router up to 7 times (in the worst case).

In most cases, a given packet will be transfered using exclusively channels from a single VN. In that context, the routing of packets is subject to the turn restrictions presented in Section 2.4.2. In this scheme, some routing options are disabled to prevent deadlocks. Following the link numbering, links (and their associated channels) are always allocated in increasing order. Following notations of Definition 2.1.3, and defining $\mathrm{Number}(\mathrm{Channels}(p))$ as the number associated to the links of channels currently owned by the packet $p$, $\forall p \in \mathcal{P}$, $\mathrm{Number}(\mathrm{Channels}(p)$ is in strictly increasing order. Therefore, $\mathrm{Next}(p_i) \in \mathrm{Channels}(p_j)$ implies that $\mathrm{Number}(\mathrm{Next}(p_i)) < \mathrm{Number}(\mathrm{Next}(p_j))$. As a consequence, $\mathrm{Next}(m_j) \notin \mathrm{Channels}(m_i)$. This reasoning extends naturally to multiple deadlocks as well.

By construction, packets from different VNs do not interfer and therefore cannot generate a deadlock. In effect, they do not share the same buffer set, and common resources such as crossbars and links follow a time multiplexing approach, which guarantees the isolation between packets of different VNs.

Following Variant B and C algorithms, packets may be exchanged between two different VNs or turn restrictions may be overpassed through the utilization of a VS buffer. In that case, we claim that packet cannot be involved in a deadlock, based on VS behaviour. In effect, VS waits until all flits of a packet $p$ are stored before re-injecting them to the crossbar. As a consequence, channels owned by $p$ are either all in the former VN or all in the new VN, and there is no dependency between the former and the new VNs. The case of VS channels exhaustion is also addressed in Section 2.4.3, and does not create deadlocks.

Regarding livelocks, a more detailed analysis is compulsory. In effect, each routing algorithm avoids livelock in a different fashion. For Variant A, VN link numbering guarantees that any packet cannot cross the same link twice. At worst, packet may be routed along all links of the interconnect, but eventually link numbering will force its routing to end. In Variant B, livelocks are avoided through RS in a straightforward manner. In effect, RS forbids to route a packet to a previously visited router. As a consequence, a packet might visit all routers in the interconnect, but eventually its routing will cease. In the case of Variant C, the packet route $R_p$ and the echoed node set $\mathrm{Echo}_p$ provide the livelock freedom guarantee. In effect, routing a packet to a router $n$ in Hierarchy mode requires that both $n \notin \mathrm{Traversed}(R_p)$ and $n \notin \mathrm{Echo}_p$. In Echo mode, routers which are listed in the echoed mode set $\mathrm{Echo}_p$ are not allowed neither. As a consequence, packet routing will terminate at worst after all nodes of the interconnect have entered the echoed node set.

### 2.7.4 Termination

Based on their safety properties, the termination of the proposed algorithms can be proved. However, some assumptions are necessary to guarantee that no packet is blocked in the interconnect due to causes unrelated to the routing algorithm (e.g. process variability, router implementation).
**Assumption 2.7.1** (Router fairness)**.** *No packet will wait infinitely to be transmitted by a router, i.e. no starvation.*

[57] Virtual Network
[58] Router Stamping

**Assumption 2.7.2** (Link delay)**.** *The time for a packet to traverse a link is bounded.*
**Property 2.7.1** (Termination)**.** *Based on Assumptions 2.7.1, 2.7.2 and the livelock freedom property detailed in Section 2.7.3, the time that each packet spends in the interconnect has an upper bound.*

Since no packet can be involved in a livelock, the number of hops it makes is bounded. Additionally, since each hop requires a bounded amount of time in our deadlock-free routing algorithms, the total amount of time has an upper bound, and algorithms do terminate.

### 2.7.5 Variant C route discovery and network partition detection

The Variant C algorithm has 2 specific properties. First, it is able to find a route from a given source node $s$ to a target node $t$, if at least one route exists. Second, if there is no route, the source node $s$ is notified that the destination is unreachable. Unfortunately, these properties are subject to a limitation on the traffic conditions.

Following the discussion in Section 2.4.3, packets transmitted using Variant C may be dropped in order to preserve deadlock freedom, if VS channels are exhausted. While experimental results show that this event is extremely rare for low traffic rates and most traffic patterns, this situation has a significant impact for highest traffic rates with light fault rates. As a consequence, following properties only hold for limited traffic rates.
**Property 2.7.2** (Route discovery)**.** *Under limited traffic, the Variant C algorithm guarantees that the packets will reach the destination ($n = t$, line 5), in the presence of any random pattern of multiple nodes and links faults, if a route exists. This property can be validated for the algorithm presented in Algorithm 3, for every source - target node pair $(s, t)$.*

Given a source node $s$ and a target node $t$, we show that under limited traffic, if there is a valid route $R$ from $s$ to $t$, the packet will be routed successfully. In effect, the only alternative is that the Variant C routing requests Echo Mode at $s$. Since Echo mode only occurs once all eligible neighbor nodes have already been visited, if this scenario were to happen, then all neighbor nodes would have been visited first, including the first router traversed by $R$. Recursively, all routers traversed by $R$ would have been visited. Eventually, $t$ would have been visited too, that is, the packet would have been transmitted to its target node at first.
**Property 2.7.3** (Network partition detection)**.** *Based on the safety and termination properties of Variant C algorithm, it can be shown that when there is no route towards the destination (i.e. network is partitioned) and limited traffic, the packet eventually returns to source node. Because this is the last option left, it attempts to use Echo mode again ( line 14), and routing is stopped. Therefore, the conclusion can be made that the network is partitioned.*

## 2.8 Conclusions

In this chapter, we proposed a set of fault-tolerant routing algorithms for 2D-Mesh NoCs[59]. These algorithms do not utilize *routing tables*, which will have a big impact on scalability of multi-core chips. Depending on the reliability target and silicon budget, any of the presented algorithm may be utilized. The Variant C algorithm presented in Section 2.4.4 offers the highest fault tolerance and supports strong properties such as partitioning detection. This algorithm is able to increase the saturation throughput under 20% of defects by a factor 10 compared to the baseline algorithm (Variant A), following Figure 2.28. A fault recovery technique is also presented in Section 2.4.1, in order to maintain the functionality of the interconnect despite the occurrence of multiple permanent and transient faults.

[59]Network on Chips

Deadlock freedom of the solution is guaranteed by the use of 2 VNs[60] and RS[61]. A VS[62] buffer offers sufficient adaptability to support the high fault tolerance while maintaining deadlock freedom. Finally, the Echo Mode guarantees that a route will be discovered, even under complex fault patterns.

In addition, the figures show that for our solution, the average latency increases with the number of nodes. In Section 2.5.1, we propose Explicit mode to overcome this issue. For streaming applications, which require large messages between few different source - target node sets, the Explicit mode limits traffic increase in case of faults. Thus, the average latency is improved significantly, based on the Variant C fault-tolerant routing algorithm.

[60] Virtual Networks
[61] Router Stamping
[62] Virtual Source

# Chapter 3

# Techniques for the self-adaptivity of parallel applications in unreliable many-core processors

The second main contribution of this thesis is a set of techniques offering fault tolerance and energy efficiency to parallelized applications running on many-cores processor chips, in the presence of large variability and frequent defects. These works led to 3 publications in international conferences of the *IEEE*. In 2010, the fault-tolerant approach was published at the *IEEE International Symposium on Network Computing and Applications* in [90]. The variability aware technique was defended in 2 papers at the *IEEE International On-Line Testing Symposium* in 2011 ([8]), and 2013 ([91]).

## 3.1 An introduction to applications execution on many-cores chips

There is currently a tremendous effort for the efficient utilization of chips consisting of hundreds and thousands of computing cores. This enthusiasm is mainly caused by the high potential of these architectures on one hand, and the large efficiency gap of current solutions on the other hand. Researchers investigate both architectural improvements and software mechanisms, which leads to a large number of varied approaches. Nevertheless, the execution of applications essentially requires three main features. The application mapping discussed in Section 3.1.1 selects the resources for each applications' tasks. The task scheduling introduced in Section 3.1.2 is responsible for defining and handling the timely execution of each task. At last, the implementation of tasks' communications and memory access are discussed in Section 3.1.3.

### 3.1.1 Application(s) mapping

The exploitation of many-core processors generally requires to divide the workload of applications into tasks, through parallelization (c.f. Section 1.1.1). In effect, a monolithic application would exhaust one computing core, while there are plenty of under-used cores in the processor. As a consequence, software developpers -aided by various tools such as compilers and profilers- have to parallelize their application before its execution on the processor.

The mapping of a parallel applications essentially consists in allocating the required resources to all applications' tasks. An illustrative example of mapping is given in Figure 3.1. Given a set of 3 applications $\{A_1, A_2, A_3\}$ on one side, and a processor consisting of $3 \times 3$ cores, the mapping process

associates each task to a processing node. From this perspective, a valid mapping has to guarantee that each and every task is associated with one and only one processing node.

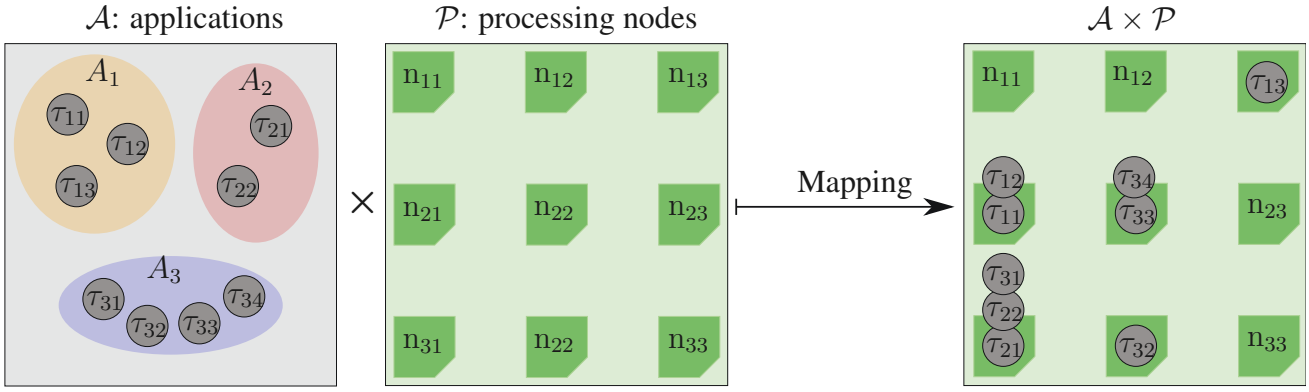



Figure 3.1: An example of application mapping to a processor chip

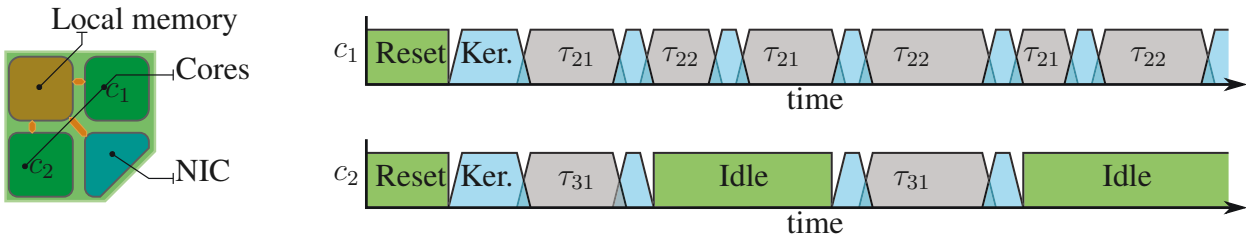
Unfortunately, mapping tasks at random -as in Figure 3.1- poses dramatic performance issues in general. First of all, nodes have a limited computation capacity (i.e. number of operations per time period), thus tasks may not be executed fast enough for the application to fullfill its requirements. Second, tasks belonging to the same application are inter-dependent. They generally require some synchronisation and/or data exchange between them. Thus, the overall application efficiency would be higher if the dependency between tasks was considered during mapping, in order to minimize the energy and latency incurred by communications and synchronizations. Third, with latest technology processes, multiple characteristics varies significantly from one node to another, within the same die. Commonly, the manufacturing variability and the temperature modify largely the computing capacity and energy efficiency of computation nodes.

As a consequence, mapping applications to future many-core chips poses serious challenges. First of all, computation of the optimal mapping has heavy computational requirements, which hardly meet the acceptable overhead for on-line application management. Conversely, off-line mapping is generally inappropriate because several decision parameters change with time, and die manufacturing and use history. As a consequence, there is a rich litterature of mapping approaches referred to in Section 3.2. The most common approach is therefore to precompute some task information off-line, and rely on a run-time framework to make an educated guess about which node shall be used for which task. To the point, [92] produces an overview of frameworks that handle Multi Processor SoC[1] platforms. This article presents 3 trade-off directions. First, the specificity of addressed applications, then the amount of hardware requirements, and finally how distributed is the framework.

### 3.1.2 Tasks scheduling

The scheduling of tasks is responsible for deciding which task should be run at any given time. In effect, each processing node has limited computing resources, as known as **cores**. Figure 3.2a depicts the main components of an example node. Within each node, a low-latency interconnect -usually a customized bus- is utilized for linking the memories, the cores and the NIC[2]. In this node, two computing cores share a local memory. A NIC is utilized for communications with other nodes. In actual designs, the number of cores fluctuates from one up to sixteen cores. Furthermore, memory hierarchy is much more complex, since up 3 levels of caching are inserted at node level. However, an high-level view of nodes architecture is considered, for the sake of genericity and readability.

[1] System on Chip
[2] Network Interface Circuit




(a) Scheme of a processing node

(b) Chronogram of the cores state over time

Figure 3.2: An example of processing node scheduling

The chronogram given in Figure 3.2b illustrates the utilisation of the presented node. In this example, 3 tasks $\{\tau_{21}, \tau_{22}, \tau_{31}\}$ have been allocated to this node. After the node has been reset, the kernel of the OS[3] is launched on each core. This OS code is responsible for critical operations such as node initialization and tasks scheduling. Then, a first task is selected by each core, and executed. In practice, switching from a task to another requires a few clock cycles in order to save the processor state (a.k.a. context) of the previous task, and restore the one associated with the new task.

### 3.1.3 Communication and memory access

In parallel applications, the access and exchange of data by tasks is a sensivive issue. In effect, all processing nodes cannot access to all memory elements with a low latency, due to physical constraints. As a consequence, inter-task communications largely depend on the chosen processor architecture and low-level software layers. In this respect, there are essentially two types of processor architectures. shared memory processors provide mechanisms to emulate the capacity to access all memory elements from all nodes. message passing architectures require that software tasks explicitly specify the amount of data and the target task. In essence, the shared memory processors require more effort to designers for providing cache coherency to the nodes, as discussed in Section 2.1.2. On the opposite, message passing processors demand additional efforts from the application developpers, since they have to explicitly describe the data streams and synchronization operations.

At the moment, both architecures have been experimentated in the litterature. For a limited number of cores, the shared memory approach has been chosen for most chips, since ease of programming is a leading factor for the adoption of multi-core processors. However, many-cores are dominated by message passing approaches, because the efficient implementation of cache coherency for such chips is extremely difficult.

## 3.2 State of art

### 3.2.1 Application models

In the litterature, message passing applications are often modeled by the directed graph of their tasks. In these graphs, vertices represent computation tasks and arcs depict communications from one task to another. The popularity of this representation stems from its readability and the explicit display of tasks proximity, in terms of tasks interaction.

[3]Operating System

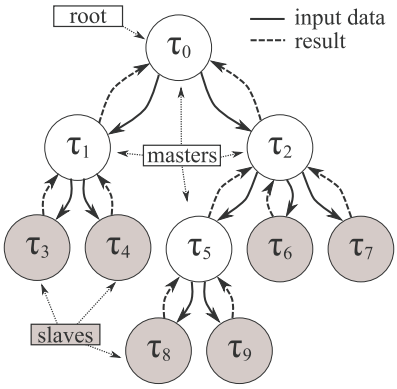



Figure 3.3: A sample hierarchical master-slave application

Amongst existing applications, many may be in fact represented by a task DAG[4] since their task graph does not include any cycle. In particular, most streaming applications can be modeled as a DAG, as shown in [93].

In particular, there exists a rich class of applications described by a DAG structure conforming to the master-slave pattern [7]. This pattern is one of the simplest ways to parallelize an application and the most popular in practice. Generally, a master task decomposes its work into smaller tasks, distributes these tasks among a farm of slaves and waits for partials results [94]. Each slave performs its processing on the received data, then returns the result of the processing back to its master, which gathers and assembles the partials results in order to produce the final result of the computation.

To overcome the centralized master bottleneck, a hierachical master-slave pattern is taken into account [95], as depicted in the illustrative example of Figure 3.3. In this model, the master at the top of the hierachy is called the root task. The root partitions the input data to the underlying masters, and so on, until the slaves are reached and directly process the reveived input data.

Finally, fork-join DAG applications structure ([96, 97]) is similar to hierarchical master-slave DAG application, but imposes more constraints onto tasks execution. In effect, fork-join DAG is a restricted form of master-slave DAG where each task is decomposed into a *fork* and a *join* operation, alike in series-parallel DAGs. In this scheme, each task first executes its *fork* operation to instantiate input data of its slaves. Then, messages are sent to slaves and *join* operation is called once all slaves returned their results.

### 3.2.2 Mapping for NoCs

The mapping of applications on NoC[5]-based processors is currently a hot topic in the academic world. For instance, An incremental strategy to map application on NoCs plateforms with multiple voltage levels is proposed in [98]. Their approach is based on a centralized global manager where a near-convex region selection is used. In [99], it is proposed to map dynamically tasks on a NoC-based processor with heterogeneous processing nodes. The mapping process is centralized onto a master that chooses where to map tasks following a predefined strategy. Authors have experimented five strategies from which *Nearest Neighbor* and *Path Load* shown better execution time and NoC utilization. Authors from [100] propose an energy-aware task mapping algorithm improving the proposition of [101], but is still centralized. In [102], authors suggest the so-called "dynamic spiral mapping" technique and target a chip with 2D-Mesh interconnect, but without taking into account the consumed energy. Authors from [103] proposed a decentralized run-time algorithm to map a task set onto an homogenous

[4]Directed Acyclic Graph
[5]Network on Chip

multiprocessor based on a network-on-chip. In [104], authors propose a run-time agent-based technique to map an application on many-cores processor chip. Based on experimentations, the proposed technique is shown to be efficient in terms of mapping computation effort and overall network traffic reduction. However, for the targeted processor chips consisting of hundreds of cores, the manufacturing variability may significantly degrade the mapping efficiency.

### 3.2.3 Variability awareness

Efficient variability-aware techniques generally require on-line mechanisms, since the effects of variability change from one die to another. However, several works propose to improve efficiency under variability by observing statistic properties of the dies. For instance, [105] presents a design-time mapping and a scheduling algorithm, based on the expected variance of processor variations.

The modelling of variability is complex and generally tightly linked with the manufacturing process and the design methodology. However, the VARIUS model proposed in [106] provides a comprehensive model for the assessment of variability effects and is widely used in the literature (e.g. [107]). In this model, a variation map is superimposed with the processor floorplan, and the characteristics (e.g. frequency, leakage) are computed with third-party tools.

Recently, the VARIUS model has been used for the core allocation within an 80-core TeraFLOPS processor in [108], and for the optimization of the voltage-island partitioning of thousand-core chips in [109]. In [110], different variation-aware mapping and scheduling algorithms were proposed. While the latter article provides interesting results for a 20-cores CMP, the solution may not suit well chips consisting of hundreds of cores with sparse cache memory.

However, the VARIUS model requires speculating on the systematic and random variations for the considered process and design. This is a key concern for the future technologies such as 22nm and 14nm, where process evolutions are expected to lead to significant changes in the variability causes and effects.

For instance, [111] claims that the systematic variation should disappear thanks to improvements in the lithography and the mask generation technology. In addition, the random variation is likely to increase significantly, due to the transistors' channel extreme shrinking. As a consequence, many low-level solutions to variability have been proposed in the literature, such as body biasing and timing faults detection mechanisms (e.g. [112, 45]). Since the adoption of these techniques and the actual evolution of manufacturing processes are uncertain, predicting the impact of variability on future designs is now a very complicated problem.

### 3.2.4 Reliability awareness

[113] proposes a set of online mapping and scheduling techniques, and an application-specific synthesis method, in order to optimise the MTBF[6] of processing nodes. In this work, authors stress the impact of temperature on various defect mechanisms such as electro-migration, thermal cycling or stress migration.

In works of Collet and Zajac [114, 115], an high-level approach to fault-tolerant chips is given. In these works, one or more specialized processing nodes, named IOPs, are designed for robustness and communication with the off-chip memory. These priviledged nodes have the responsability for dispatching tasks to other nodes, following a fault-tolerant nodes discovery. The detection and correction

[6] Mean Time Between Failure

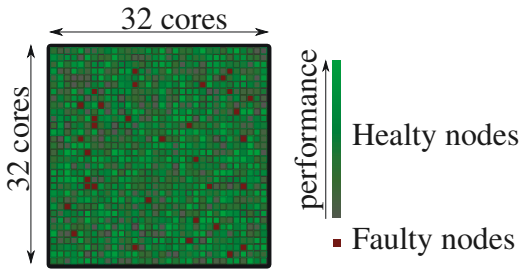



Figure 3.4: Schematic view of a target many-cores processor

of node faults is based the redundant execution of tasks. In this approach, all tasks communicate directly with IOPs, which may incur communication congestion for large chips.

Authors from [116] propose a design-time multi-objective strategy for mapping and routing in NoCs, to increase the robustness. However, only link faults are considered.

## 3.3 Context

### 3.3.1 Motivations

The advent of future nanoscale technologies opens the way for hundreds or thousands cores on a single die, but many obstacles remain for whom projects to utilize them. In particular, the reduced size of components built with aggressive technology, combined with the manufacturing variability, leads to a larger heterogeneity on transistors' parameters, resulting in very disparate specifications and potentially decreasing the chip reliability. Thus, the utilization of an unreliable many-core processor similar to Figure 3.4 has two main consequences. Firstly, due to the restricted node's memory size and the amount of exchanged messages, every node cannot have a global view of the whole system. Instead, it may just have a partial and local view of its nearest neighbors. Secondly, nodes, links and routers may disappear at any time because of defects, and might reappear later. Overall, future multi-core processors will require fault-tolerant techniques to mitigate permanent and transient faults, and be aware of specification heterogeneities.

In order to ensure that all applications on the system will return the expected results, despite complex combinations of hardware defects, the synergy of techniques at the technology, hardware or software level is required. In particular, this target requires to handle the defect of processing node while they execute applications' tasks. As a consequence, the aim of this chapter is first to deal with permanent faults at the task and OS[7] levels, relying upon a fault-tolerant hardware, such as a core interconnect equipped with a fault-tolerant routing algorithm proposed in Chapter 2. Second, the effects of nodes variability are mitigated with a variability-aware mapping technique.

Second, the proposed solution has to respect applicative constraints as much as possible. In particular, in the absence of new defects, applications should exhibit performance comparable to fault-free approaches. While off-line approaches inherently outperform on-line techniques when the applicative scenario is known in advance, a self-adaptive approach is able to equal or even overpass them in the presence of variability and unknown applicative use-cases.

The dynamic task mapping aproach presented in this chapter follows the two objectives cited above in a generic manner. After defining compliant applications architecture in Section 3.4, a fault-tolerant

[7]Operating System

mapping (and re-mapping) solution is proposed in Section 3.5. This solution is improved in Section 3.6 to handle more efficiently processors subject to a large variability.

In order to be efficient, a system level fault-tolerant strategy requires some low-level technique to capture fine grain behavior and to react quickly enough to defects. In the targeted architecture, the NoC[8] routing too must be fault-tolerant. Therefore, the proposed strategy relies upon an adaptive routing algorithm, which can adapt the route of messages to the network degradations. Additionally, the routers, links and computing nodes must feature run-time diagnostics and identify defective parts. This requirement is not covered in this contribution, but several techniques are discussed in Section 2.1.4.

### 3.3.2 Formalism

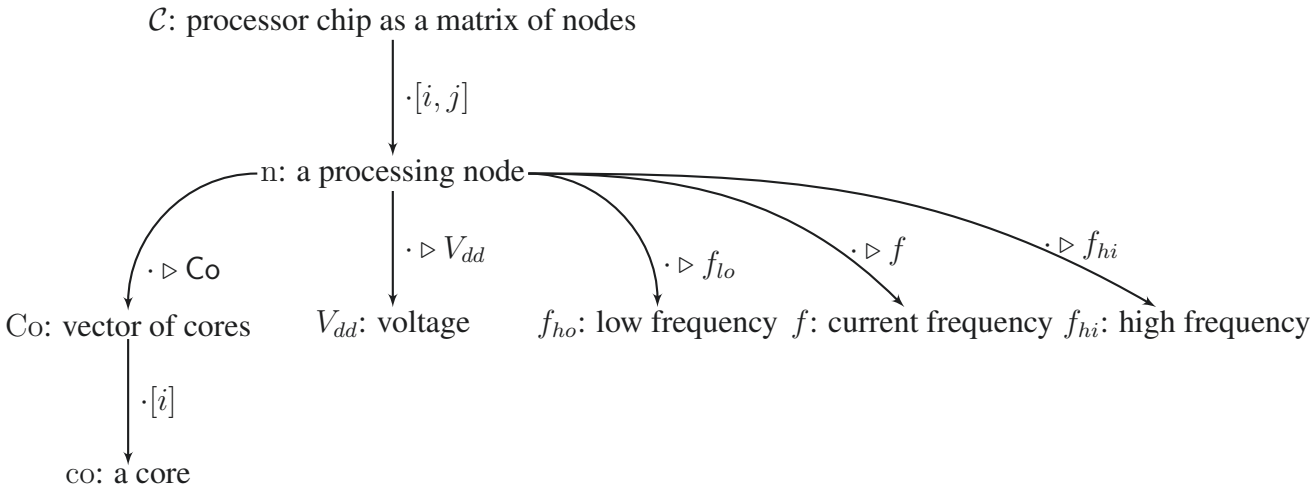



Figure 3.5: Formal structure of the considered processor, with objects name and relationship

Given a processor $\mathcal{C}$ consisting of $W \times W$ processing nodes connected by a 2D-Mesh NoC, the mapping problem requires to allocate sufficient processing resources to all applications $\{A_1, A_2, ..A_i\}$ of the applications set $\mathcal{A}$. The computation of each application $A_i$ is divided in a set of tasks $\{\tau_{i1}, \tau_{i2}, ..\tau_{ij}\}$, following Figure 3.1. In addition, the mapping relation is defined in Definition 3.3.1.

**Definition 3.3.1** (Mapped tasks)**.** *Given a computation node $n$, the mapped task set* $\mathrm{Mapped}(n)$ *contains all tasks that are currently mapped to $n$.*

### 3.3.3 Models

#### Processing nodes

In the context of many-cores processors, the GALS[9] architecture appears to be very promising [117]. In this approach, asynchronous interconnection networks provide a flexible communication channel between independent processing nodes, with the advantage of near-zero idle power consumption and reduced sensitivity to variability. Fine-grained voltage islands and DVFS[10] have been integrated to this methodology naturally in [118], and unlock a high energy efficiency by scaling each prcessing node at the optimal frequency and voltage, depending on the application needs.

[8] Network on Chip
[9] Globally Asynchronous Locally Synchronous
[10] Dynamic Voltage Frequency Scaling

In this contribution, processing nodes consist of one or more processing cores $(c_1, c_2, ..c_k)$, following Figure 3.6. A NIC[11] is responsible for communications towards other nodes. Following the GALS approach, each node has an independent clock generating circuitry.

Since the considered chips are exposed to high variability and defect rates, nodes are equiped with multiple variability and temperature sensors. Nodes additionally embed BIST[12] circuitry to detect permanent faults and calibrate nominal frequencies. Transient and intermittent faults are detected through the use of double latch techniques such as [45].

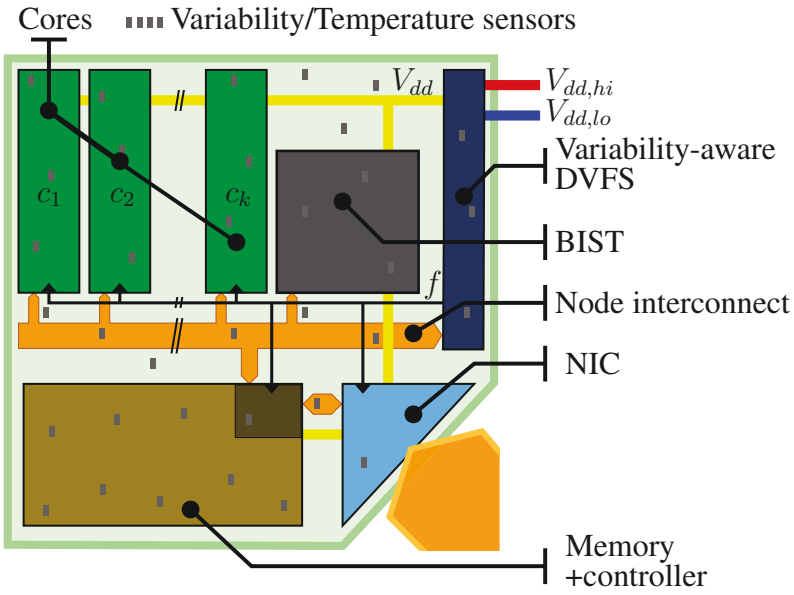



Figure 3.6: Scheme of the target processing node

The node's DVFS circuit brings adaptivity at the node level. When the node is not used, it is completely deactivated using power-gating. In that case, the energy consumed by this node is neglected. Utilized nodes use fine-grained DVFS as proposed in [4, 119]. In this approach, 2 voltages -$V_{dd,hi}$ and $V_{dd,lo}$- are distributed to all nodes. The DVFS circuit rapidly switches from one voltage to another (i.e. $V_{dd}$-hopping), and frequency is adjusted for each voltage with a minor delay, respecting variability measures of the node's sensors. The maximum frequency at $V_{dd,lo}$ (respectively $V_{dd,hi}$) is denoted $f_{lo}$ (respectively $f_{hi}$). The resulting voltage ($V_{dd}$) and frequency ($f$) are distributed to all circuits of the node, including cores, NIC and memory.

In our proposition, a trivial task scheduling model is considered. We consider that tasks are executed with a non-preemptive round robbin scheduler, and cores share tasks without significant overhead. Following this assumption, the number of cores per node does not affect our works, since only the overall computation capacity of the node matters.

The considered interconnection network is a 2D-Mesh NoC. The impact of variability on the interconnect is not considered, since the litterature already suggests several mitigation techniques in Section 2.1.4. Moreover, we consider that the interconnect was scaled appropriately for the application, thus contention infrequently occurs in the interconnect. Finally, the assumption is made that communications to other nodes are directed by the memory controller independently from cores. As a consequence, communications do not require the control of a processing core, which are therefore available for other tasks.

[11] Network Interface Circuit
[12] Built-In Self-Test

### Variability

For decananometric technologies, chips variability will be a major issue. The process variations mostly affect the transistors' threshold voltage $V_{th}$. The variability effects are usually categorized into systematic and random variations. Both random and systematic variations follow additive independent normal distribution of 0-mean and standard deviation of respectively $\sigma_{rand}$ and $\sigma_{syst}$. In addition, the systematic variation of two different transistors is correlated by a distance-dependent factor $\rho$, as shown in Equation 3.1, based on the $V_{th}$'s average, denoted $\overline{V_{th}}$.

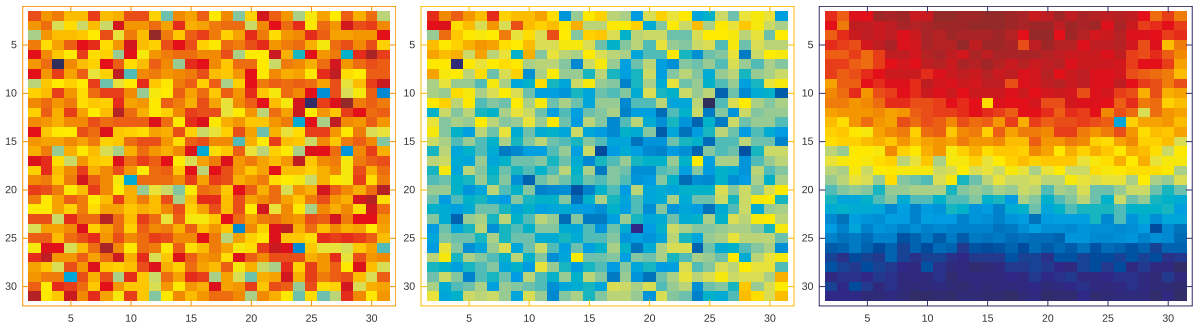
$$V_{th} = \overline{V_{th}} + \mathcal{N}(0, \sigma_{rand}) + \mathcal{N}(0, \sigma_{syst}, \rho) \tag{3.1}$$

The proposed variability model is based on the VARIUS model ([106]), yet our model is not specific to a given design floor planning, at the expense of decreased accuracy. Instead, the generic processor floorplan shown in Figure 3.4 is used. In the context where there is a large uncertainty about future etching processes and low-level design standards, this approach provides a qualitative insight of future chips behaviour. This approach is further detailed in Appendix A.1.

| Scenario | Context | $\sigma_{rand}/\overline{V_{th}}$ [%] | $\sigma_{syst}/\overline{V_{th}}$ [%] |
|---|---|---|---|
| Case 0 | Variability is neutralized | 0 | 0 |
| Case 1 | Variability is stationary | 6 | 6 |
| Case 2 | 2x random variation | 12 | 0 |
| Case 3 | 4x random variation | 24 | 0 |
| Case 4 | High random and systematic variations | 24 | 12 |

Table 3.1: Variability scenarios

In Table 3.1, five variability scenarios are considered, based on the literature. In effect, depending on the manufacturing process characteristics and design methodology, the impact of variability will differ substantially. In the first scenario (Case 0), the variability is cancelled by CMOS technology improvements and variation-aware design flow. In Case 1, the variability does not evolve, thanks to manufacturing and design practice improvements. In Case 2 and 3, the random variation increases as a result of transistors' channel shrinking, but the systematic variations disappear due to the advances in the lithography and mask generation. Lastly, Case 4 represents the situation where both random and systematic variations are important.



(a) $\sigma_{syst}/\overline{V_{th}} = 0$ (b) $\sigma_{syst}/\overline{V_{th}} = 6\%$ (c) $\sigma_{syst}/\overline{V_{th}} = 12\%$

Figure 3.7: Impact of the systematic variations on the cores frequency

While the random variations are considered as the most important variation issue, the systematic variations may be a concern in the future as well. In effect, manufacturing constraints in the decananometric technologies may change substantially in comparison with current CMOS processes. Figure 3.7

represent the frequency of computation nodes for an hypothetical processor of $32 \times 32$ cores, with various rates of syystematic variation. In particular, when the systematic and random variations are of the same order, several regions may appear, as shown in Figure 3.7. In this context, some regions mostly consist of efficient nodes and conversely other regions contain many ineffective nodes.

### Energy

The energy consumption essentially results from computations at processing nodes, and communications in the interconnect. The computational energy is mainly affected by the voltage and frequency of the processing nodes. Besides, communication energy is strongly correlated with the duration of the transmission, the number of required hops and the size of the messages payload.

For a target frequency $f_t$, the controller of node $n$ will adjust the voltage duty cycle to obtain the desired operating frequency in average. The time ratio $\theta_{lo}$ (respectively $\theta_{hi}$) for low voltage operation at $V_{dd,lo}$ and high voltage $V_{dd,hi}$ are proposed respectively in Equation 3.2 and Equation 3.2.

$$\theta_{lo} = \begin{cases} 1 & \text{if } f_t \leqslant f_{lo} \\ \frac{f_{hi} - f_t}{f_{hi} - f_{lo}} & \text{if } f_{lo} < f_t < f_{hi} \\ 0 & \text{if } f_{hi} \leqslant f_t \end{cases} \tag{3.2}$$

$$\theta_{hi} = \begin{cases} 0 & \text{if } f_t \leqslant f_{lo} \\ \frac{f_t - f_{lo}}{f_{hi} - f_{lo}} & \text{if } f_{lo} < f_t < f_{hi} \\ 1 & \text{if } f_{hi} \leqslant f_t \end{cases} \tag{3.3}$$

The instant power consumed by the node $n$ is given in Equation 3.4, where $\alpha_1$, $\alpha_2$ and $\alpha_3$ are fitting parameters enclosing the effects of the gates activity factor, global capacitance and additional parasitic effects.

$$P(n, f) = \underbrace{\alpha_1 \cdot V_{dd} \cdot e^{\alpha_2 \cdot V_{dd}}}_{\text{leakage}} + \underbrace{\alpha_3 \cdot V_{dd}^2 \cdot f}_{\text{switching}} \tag{3.4}$$

The average energy per operation cycle for a node $n$ running at target frequency $f_t$ is given in Equation 3.5. When operations are requested on a node (i.e. $f_t > 0$), the node uses DVFS. Otherwise, power-gating is used when no computation is required (i.e. $f_t = 0$).

$$E_{op}(n, f_t) = \begin{cases} \frac{\theta_{lo} P(n, f_{lo}) + \theta_{hi} P(n, f_{hi})}{f_t} & \text{if } f_t > 0 \\ 0 \text{ (power-gating mode)} & \text{if } f_t = 0 \end{cases} \tag{3.5}$$

In Figure 3.8, a qualitative view of the evolution of node energy depending on the target frequency is given. Four phases appear in this figure. First, for zero-frequency target, no energy is consumed, thanks to power-gating. Second, for $f_t < f_{lo}$, the energy per operation decreases rapidly, since the node steadily functions at $f_{lo}$ frequency, and wastes unused cycles with idle operations. Third, the energy increases in the DVFS range, since $f_{lo} P(n, f_{hi}) > f_{hi} P(n, f_{lo})$. Finally, for frequency targets beyound $f_{hi}$, the controller does staturate the output frequency, hence energy stabilizes, but application would suffer from unexpected latencies.

Regarding the interconnection network, the consumed energy is roughly proportional to the product of message's payload and the message utilization of the interconnect resources. The latter is either

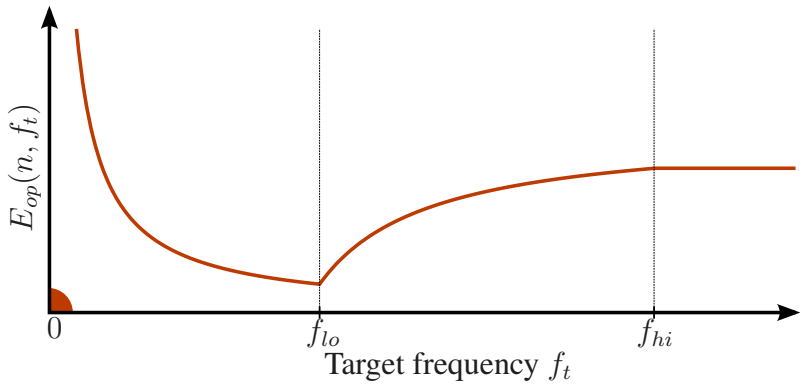



Figure 3.8: Energy per operation, as a function of the node's target frequency

measured as the *one-way* message latency, or the number of message hops. Under the assumption that there is few contention within the routers, both metrics are strongly correlated anyway.

Followingly, the energy consumed for the transmission of one flit is given in Equation 3.6, where $d_M(n_i, n_j)$ is the Manhattan distance between the node $n_i$ and node $n_j$. $\alpha_4$ includes the energy required for the flit reception and storage within the router, the crossbar arbitration and the emission of the flit over the next link. In practice, message might take detours, following interconnect defects or congestion, and overall energy would therefore increase.

$$E_{flit}(n_i, n_j) \quad \geqslant \alpha_4 \cdot d_M(n_i, n_j) \tag{3.6}$$

## 3.4 Programming model for fault tolerance

The programming model is critical for the application fault tolerance. In effect, faults may strike any task at any time, through one of the many fault mechanisms. Hence, detecting the fault and more importantly returning to a valid application state are two major challenges. In this aim, the selection of a compliant application model is a crucial choice. Firstly, a resilient task organization is proposed in Section 3.4.1, and essential fault tolerance mechanisms are explained. Second, the expression of application requirements is discussed in Section 3.4.2, highlighting the genericity of our model.

### 3.4.1 Tasks organization

In this contribution, the fork-join DAG[13] model is used for the organization of tasks. In effect, this scheme supports well the detection and recovery of faults. In that scheme, each task $\tau$ has one parent task denoted $\tau \triangleright \mathsf{Pa}$, except the top hierarchy task, which is named as **root task**. Optionally, tasks may have one or more children tasks, which set is denoted $\tau \triangleright \mathsf{Ch}$. The knowledge of the application execution flow *a-priori* is not required, hence tasks may create new children dynamically at run-time. During its creation, essential task characteristics are required, as detailed in Section 3.4.2. The children and grand-children sets are defined in Definition 3.4.1

**Definition 3.4.1** (Grand children set)**.** *Given a task $\tau_p$, the grand children set $\mathsf{Grand}(\tau_p)$ contains all tasks $\tau_c$ which are $\tau_p$'s children, either directly (i.e. $\tau_c \in \tau_p \triangleright \mathsf{Ch}$) or indirectly ( i.e. $\tau_c \in \tau_p \triangleright \mathsf{Ch} \circ \ldots \circ \triangleright \mathsf{Ch}$).*

[13] Directed Acyclic Graph

In this approach, the intrinsec task hierarchy is a major advantage towards fault tolerance. In effect, each parent task is made responsible for the correct execution of its children. The fault detection is achieved by exchanging IAM[14] between parent and their children tasks. When a task lacks an IAM from an interacting task (either parent or child), the latter is considered faulty. In case the parent task is faulty, children will terminate by themselves, effectively freeing computation capacity for other tasks. If the faulty task is a child, the task is re-mapped to another resource, as discussed in Section 3.5. Since applications naturally support the creation and suppression of tasks at run-time, most mechanisms for the task re-execution are already included.

In a global view, the set of applications running on the processor is denoted $\mathcal{A}$, according to Figure 3.9. Each application is further divided into tasks, generically named $\tau$. Besides the parent task and the list of children tasks, each task has a set of requirements Req detailed in Section 3.4.2, and is associated with its execution binary code Code.

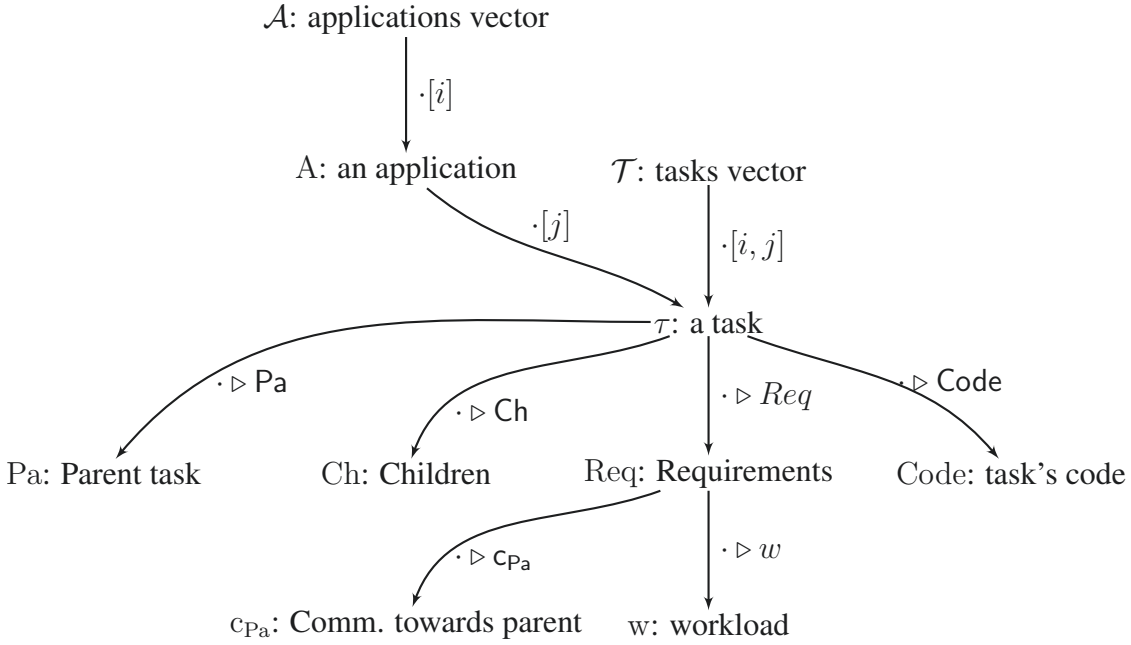



Figure 3.9: Formal structure of the applications and tasks, with objects name and relationship

### 3.4.2 Task requirements and node slack

Depending on the applicative goals and processor architecture, tasks are subject to different constraints or requirements. The proposed approach is capable of handling virtually any task-level requirement, based on the dynamic task mapping technique described in Section 3.5. Since most requirements control the utilization of shared resources in the node, our approach demands to keep track of the node slack.

In our approach, a node slack measure is opposed to each requirement. Both requirements and slacks are conceptually represented as vectors of real numbers of the same dimension denoted respectively $(r_1, r2, .., r_k)$ and $(s_1, s_2, .., s_k)$. Practically, several elements may be represented by integers or boolean in order to reduce the memory footprint of requirements. Noting $\tau \triangleright \mathrm{Req}$ for the requirements of task $\tau$ and $\mathrm{Slack}(n)$ for the slack of node $n$, the valid mapping constraint is presented in

[14] I am Alive Message

Equation 3.7.

$$\tau \triangleright \mathrm{Req} \leqslant \mathrm{Slack}(n) \quad \Leftrightarrow \quad r_i \leqslant s_i, \forall i \in [1,k] \tag{3.7}$$

To the extent of this thesis, the considered requirements are inter-task communications and workload. For the sake of readability, the requirements of a task $\tau$ are presented as the aggregation of the worload $w$ and communications $c_{Pa}$, following Figure 3.5. However, requirements could be represented by a vector $(w, c_{Pa})$ as well. Assuming that each task operation and communication of data elements require one cycle on average, the node slack is defined in Equation 3.8.

$$\mathrm{Slack}(n) \equiv \left( n \triangleright f_{hi} - \sum_{\tau \in \mathrm{Mapped}(n)} \tau \triangleright \mathrm{Req} \triangleright w \quad , \quad n \triangleright f_{hi} - \sum_{\tau \in \mathrm{Mapped}(n)} \tau \triangleright \mathrm{Req} \triangleright c_{Pa} \right) \tag{3.8}$$

The expression of application timing constraints in many-cores processors is a tedious issue. In particular, the opposition between adaptivity and guarantees discussed in Section 2.1.3 strongly limits the capacity of self-adaptive applications to provide trustable timing guarantees. In addition, the determinism of applications that utilize chips subject to high variability and defect rates is questionable. As a consequence, the proposed approach is certainly not suitable for time-critical applications. However, our contribution readily applies to soft real-time applications such as multimedia decoding. In effect, while a few deadlines might be missed following the occurence of defects, most will be honored.

In this chapter, we consider that applications execute repetitive sequences of computation on different data, each subject to a termination deadline. The derivation of task-level deadlines out of application-wide constraints is a complex process. In the context of many-core chips with processing nodes running at different frequencies. The approach described below consists in expressing deadlines as task workload per unit of time. This loose relation offers satisfying results as long as task granularity is small enough, and interconnect latency is under control. Since there is a large number of parasitic effect, a completion margin is added to the measured tasks' workload.

Let us denote $t_{fork,j}$ and $t_{join,j}$ the termination time of fork (respectively join) operation of the task $\tau_j$. In addition, $\omega(\tau)$ is the representative number of operations for task $\tau$. Note that this value shall range between average and worst-case number of operations of the task, providing a continuous tradeoff for the time-criticity of the application. The critical task may be then determined by computing the cumulated number of operations $\Omega(\tau)$ for each task $\tau$. Following Equation 3.9, the maximum number of representative instructions along all each task sequences $\Omega_{max}(\tau)$ is obtained in Equation 3.10. Finally, the workload of each task is defined in Equation 3.11, by allocating time in order for all task sequences to finish on application deadline $\Delta T$. Necessary margins are introduced by constant $M$ and optionally, by artificially increasing the representative number of tasks' operations.

$$\Omega(\tau) = \omega(\tau \triangleright \mathsf{Pa}) + \omega(\tau) \tag{3.9}$$

$$\Omega_{max}(\tau_p) = \max_{\tau_c \in \mathsf{Grand}(\tau_p)} (\tau_c) \tag{3.10}$$

$$\tau \triangleright \mathsf{w} = \frac{\Omega_{max}(\tau)}{\omega(\tau) \times (\Delta T - M)} \tag{3.11}$$

A significant strength of our approach is that additional requirements may be added to $\tau \triangleright \mathsf{Req}$ with little additional engineering. For instance, for processors consisting of heterogeneous cores, task requirements may also embed the set of requested core extensions (e.g. cryptography, support of floating-point operations) or required core type. Memory requirements may be included as well, so that the mapping

algorithm dispatches tasks in order not to overpass local memory capacity. Finally, limitations imposed by the OS[15], such as the number of tasks per node or the criticity of tasks, may be included as well.

## 3.5 Generic fault-tolerant mapping

In this section, a self-recovering strategy is presented, which is able to "re-map" dynamically application tasks on a multi-core processor, under the presence of multiple interconnect and nodes faults.

Based on lower level fault-tolerant techniques, this self-recovering strategy ensures the application termination and the delivery of expected results, despite multiple node and interconect defects in a 2D-Mesh topology. It has been demonstrated, based on a statistical analysis, that the proposed technique is able to re-map the tasks of faulty nodes with a predictable number of hops. The theoretical results have been validated by simulations. It has been demonstrated that the Motion JPEG 2000 application can be parallelized and formally represented as a DAG[16], following our approach. It worth noting that the proposed technique has been validated by the simulation of a 1000 cores system, in the presence of routers and links faults up to 10%. Therefore, the proposed technique has been shown to be efficient for seamless execution of parallel streaming applications and to provide the *Execution Time Reduction Ratio* close to ideal.

### 3.5.1 Tasks responsibilities

The relationship and commitment between parent and children tasks follow the task DAG hierarchy. The application nested structure and its unknown execution flow lead to a straightforward hierarchical organization, in which each level has its own responsibilities. On one side, each parent is responsible for its children from their mapping to their completion. On the other side, children tasks have to report to their parents periodically, either through IAM[17] or application data exchange.

Whenever a task has to create new children tasks, the process to follow mainly consists of two stages. Firstly, the required resources are found and then reserved amongst available nodes, using the mapping procedure detailed in Section 3.5.2. Secondly, the task code is physically transferred to its allocated destination. After these two stages, the newly created task is ready and is notified by its parent. Then, the parent task can send the data to be processed, and the child task execution starts. After completion, each child task sends the resulting data to its parent.

During the execution of a child task, some defects may occur. Based on the fail-silent assumption, each child task has to send periodically an IAM to notify its parent that it is still running. Conversely, the parent task verifies that its children are alive by checking the reception of IAMs. If one of these messages has not been received, the parent infers that the node currently executing the child task was hit by a defect, and consequently declares the child task as faulty. Subsequently, the master task re-maps the uncompleted child task onto another node. When the faulty child is a parent itself, all its children are lost too, and therefore all their sub-trees. In order to maintain consistency inside the application, all lost sub-trees have to be re-mapped, on the remaining fault-free nodes. As a consequence, when a child of a faulty parent task sends an IAM, an error will be returned by the interconnect, following a Nack[18] or a timeout, since its parent is unreachable, and thus task self terminates. The frequency at which the child has to send the IAM is affected by application parameters such as task duration and size of

[15] Operating System
[16] Directed Acyclic Graph
[17] I am Alive Message
[18] Negative Acknowledgement

exchanged messages. In any case, the emission occurs at least once during the task execution. Indeed, the time elapsed between two consecutive IAMs is the worst-case latency between the occurence of a fault and its detection. It is worth noting that IAMs are very short messages, and in comparison to other techniques such as *Gossiping* ([120, 121, 122, 123]), they require only a very small bandwidth.

The protection mechanism against defects appearing in the node that executes the root task is different. In effect, root tasks do not have parents by definition. In our approach, a **latent clone** is mapped for each root task on a different node. This clone task periodically wakes up only to check that the original root's node is healthy. If the original root's node fails, the latent clone starts normal operation. A latent clone is mapped to another node and children tasks are re-mapped as usual.

### 3.5.2 Fault-tolerant nearby mapping

When a task creates new children, it needs to find fault-free nodes capable of executing them. Assuming that there is no mechanism to centralize the knowledge about the system, the parent task has to scan sequentially the existing nodes, in order to check their availability. Thus, in the **nearby mapping** strategy, the parent task successively tries nodes starting from the nearest. Each interrogated node returns a cost metric that allows for self-adaptive task mapping, as discussed in Section 3.6.

The general mapping behavior is presented in Figure 3.10. The parent task sends a *Request* message to one of its nearest nodes, beginning with one-hop away nodes. Each fault-free node receiving a request will acknowledge it either with an *Accept* message, if the node supports task requirements, or *Reject* if not. Upon emission of an *Accept*, node resources are automatically reserved, and considered during further mapping requests. Similarly, if the target node has failed or is not reachable by the NoC[19] routing algorithm, the interconnect will issue an *Error* message, meaning that the target node cannot be used.

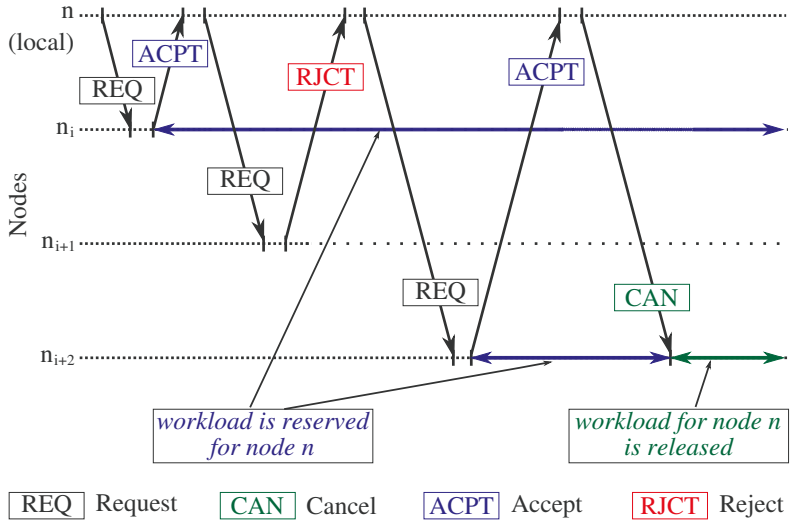



Figure 3.10: Example of exchanged messages during the mapping procedure from (local) node n.

This 2-stages message exchange is sufficient for straightforward fault-tolerant mapping. Yet, the quality of the mapping is improved substantially by testing alternative nodes after finding the first suitable node. In effect, various effects such as variability introduce node heterogeneity, as discussed in Section 3.6. Hence, Figure 3.10 actually presents a 4-stages exchange. In effect, after a first suitable node $n_i$ is found, requests are sent to other nodes. Finally, the node $n_{i+2}$ is interrogated. However,

[19]Network on Chip

the parent task prefers node $n_i$ -according to its cost measurement- and consequently returns a *Cancel* message to node $n_{i+2}$.

The criterion for mapping cost may vary largely to represent adequately the processor and applicative issues of interest. Further details are given in Section 3.6, with the presentation of variability-driven criteria. In Algorithm 5, two criteras related to cost are exploited. $\text{Cost}(n_{Pa}, n, \text{Req})$ represents the actual execution cost of a task with requirements Req on a node $n$, with parent task's node $n_{Pa}$. Accordingly, the minimum achievable cost is given by $\text{Cost}_{min}(n_{Pa}, n, \text{Req})$. The latter metric permits to stop the search earlier, under the assumption that the minimum achievable cost never decreases, during the search.

**Algorithm 5** Self-adaptive mapping of a task $\tau$

**Require:** $n_{Pa}$: the node executing the parent task
**Require:** Req: the requirements of task $\tau$
**Require:** $listToVisit$: the list of nodes to visit

```
 1: procedure GENERICMAPPING(n_Pa, Req, listToVisit)
 2:     n_best ← ∅
 3:     K_best ← ∞
        /*Compute initial minimum cost and check requirements for the local node*/
 4:     K_min ← Cost_min(n_Pa, n_Pa, Req)
 5:     if Req ⩽ Slack(n_Pa) then
 6:         n_best ← n_Pa
 7:         K_best ← Cost(n_Pa, n_Pa, Req)
 8:     end if

 9:     while K_best > K_min and listToVisit ≠ ∅ do
10:         n_i ← POP(listToVisit)
11:         (message, K) ← SENDREQUEST(Req, n_i)
12:         if message = Accept then/*Requirements are achievable.*/
13:             if K < K_best then
14:                 SENDCANCEL(n_best)/*only if n_best ∉ {n_Pa, ∅}*/
15:                 n_best ← n_i
16:                 K_best ← F
17:             else
18:                 SENDCANCEL(n_i)
19:             end if
20:         end if
21:         K_min ← Cost_min(n_Pa, n_i, Req)
22:     end while

23:     if n_best = ∅ then
24:         n_best ← n_Pa
25:     end if
26:     return n_best
27: end procedure
```

In addition, the list of candidate nodes $listToVisit$ is required by the mapping algorithm. In this thesis, this list is ordered from the nearest to the farthest. Alternative orderings may also be chosen, for instance to explore specific areas in the processor chip. Over time, the list of candidates changes to reflect the apparition of defects and the previous mapping of other tasks. In effect, each node memorizes when other nodes are unreachable (because of defects) or unavailable (because its resources are exhausted). Nevertheless, these informations are discarded after an appropriate period, since the processor dynamics render them obsolete.

Algorithm 5 begins by checking if the task requirement can be met onto the local node $n_{Pa}$ (line 5). Then, the first available node among a node list $listToVisit$ is searched. That is to say, a fault-free

node able to meet the task requirement. First, a request is sent to the next node in the list (lines 10, 11). If this node is able to meet the requirement (line 12) and has a lower cost (line 13), the currently visited node becomes the best node ($n_{best}$) (line 15) and the previous one is canceled (line 14). Otherwise, the current node is canceled (line 18). The loop on line 9 is repeated until the best cost found is lower than minimum cost achievable by remaining candidate nodes, or all nodes in the candidate list are visited. At the end, if no node was found (line 23), the task is mapped locally.

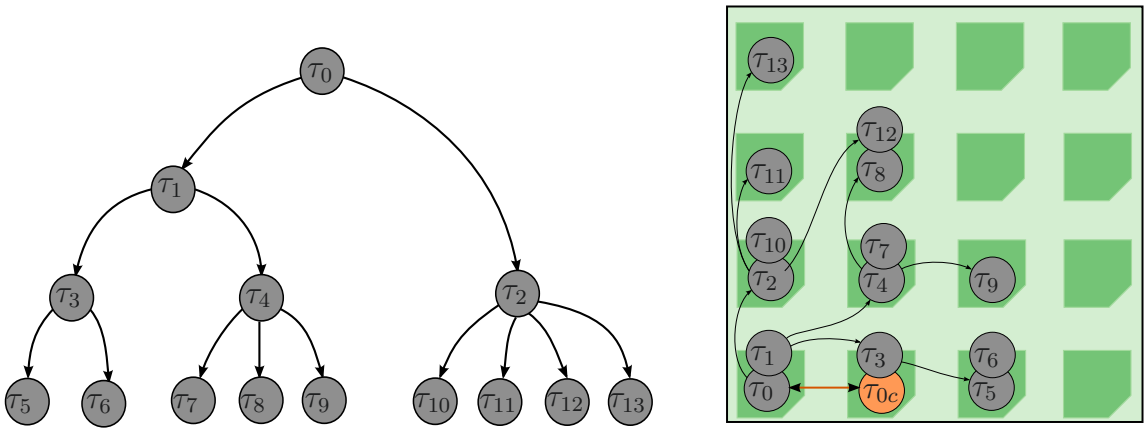
The ONREQUEST procedure presented in Algorithm 6 is executed by each node receiving a request message. First, the task requirement is checked (line 2) and the resulting quality factor is computed (line 3). Then, a message is sent back to notify whether the requirement are met (line 5), or not (line 7). If an *Accept* message is sent (line 5), the node is reserved until a *Cancel* message (Algorithm 5, line 14) has been received. Eventually, the task is started when the parent tasks effectively sends the task code and input data to the reserved node.

**Algorithm 6** Request procedure for mapping the task $\tau$ on node $n$

**Require:** $n_{Pa}$: the node executing parent task
**Require:** $n$: the current node
**Require:** Req: the requirements of task $\tau$

```
1: procedure OnRequest(n_Pa,n,Req)
2:     if Req ⩽ Slack(n) then
3:         K ← Cost(n_Pa, n, Req)
4:         Reserve task resources according to Req
5:         SendAccept(K, n_Pa)
6:     else
7:         SendReject(∅, n_Pa)
8:     end if
9: end procedure
```

In Figure 3.11, a sample application and its initial mapping on a $4 \times 4$ processor are presented. The latent clone $\tau_{0c}$ of the application's root task $\tau_0$ is affected to a different node, and children tasks are mapped iteratively following the Nearest neighbor strategy, where the first suitable node is chosen, without considering the mapping cost, i.e. $\text{Cost}_{min}(n_{Pa}, n, \text{Req}) = \infty$.




(a) Sample application (b) Initial mapping

Figure 3.11: A sample master-slave application and its initial mapping

As a consequence of defects, the distance between children and their parent will increase and degrade the system performance. However, in the case of 2D-Mesh, the number of available neighbors increases quadratically with the number of hops, as stated in Definition 1, and thus the degradation is limited.

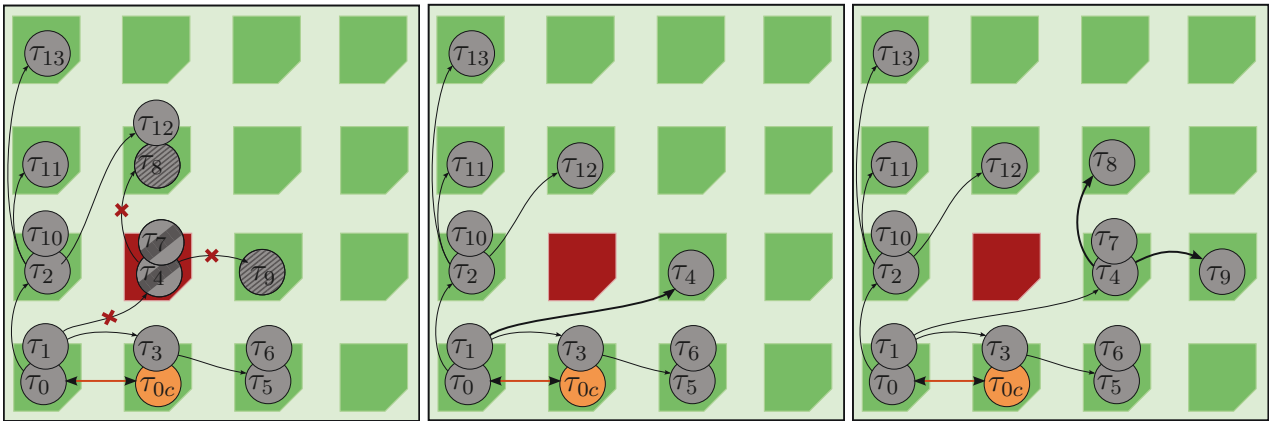
**Definition 1.** *Assuming a fault-free unbounded 2D-Mesh domain, i.e. with a large number of nodes, the nodes search within a maximal number of hops $H$ from a node $n$ at the position $(i, j)$ in each four directions is considered. Then, the amount of reachable nodes and the average number of hops for reaching these nodes are given respectively by Equation 3.12 and Equation 3.13.*

$$\text{Nodes}(H) = \sum_{i=1}^{H} (4 \cdot i) = 2 \cdot H \cdot (H+1) \tag{3.12}$$

$$\overline{\text{Hops}}(H) = \frac{\sum_{i=1}^{H} (4 \cdot i^2)}{\sum_{i=1}^{H} (4 \cdot i)} = \frac{2 \cdot H + 1}{3} \tag{3.13}$$

### 3.5.3 Mapping in the presence of faults

Figure 3.12 depicts the remapping of the application shown in Figure 3.11a mapped on a processor consisting of $4 \times 4$ computing nodes. Following the initial assignement proposed in Figure 3.11b, Figure 3.12a displays the consequences of a node fault. Tasks executed on the faulty node are lost and communication to this node are interrupted. As a consequence, the parent task $\tau_1$ will init re-mapping of task $\tau_4$. Since communications towards children tasks $\tau_8$ and $\tau_9$ are interrupted as well, these tasks are terminated by the OS[20]. Figure 3.12b and Figure 3.12b respectively show the processor's state after task $\tau_4$ and children tasks $\tau_7, \tau_8, \tau_9$ have been remapped.




(a) Node fault occurence (b) Remapping of $\tau_4$ (c) Remapping of children tasks

Figure 3.12: Partial re-mapping of an application following a node fault

In general, when new faults occur, only the part of the application relative to the defective node is stopped, since every parent task is only responsible of its own children sub-trees. The rest of the application continues its execution seamlessly. However, it must be noticed that parents close to the root -in the task hierarchy sense- show higher fault impact, since more children depend on them. Nevertheless, our technique can guarantee that in the presence of faults, the application will still run without external intervention. Specifically, the defect of the node executing the root task $\tau_0$ leads to the complete re-mapping of the application, starting from the clone of the root task $\tau_{0c}$, which enables the application to recover automously.

[20]Operating System

### 3.5.4 Analysis

In order to analyze the message complexity of one task mapping, the defect of some already mapped children during the mapping phase is not considered. Another inherent assumption is that the regarded parent task and all its ancestor are fault-free. Indeed, if one of these nodes had failed, its parent instead would process to task re-mapping. Each time a child fails, its parent has to search for another suitable node. In order to successfully map $n$ children, $n + k$ tries will be necessary, with $k$ unsuccessful tries. Assuming that no task is mapped, except the parent task, i.e. nodes are either faulty or available. Then, we are interested in finding the number of extra tries $k$ needed, such that $n$ tries are successful.

Let $X$ be the random variable giving the number of faulty nodes $k$ hit during the search of $n$ nodes, under a node reliability of $P_s$. In this contribution, the random variable $X$ follows the negative binomial distribution [124], i.e. $X \sim \mathcal{NB}(n, P_s)$. According to the negative binomial distribution properties, a probability analysis is displayed in the Table 3.2. For each fail-free probability ($P_s$=0.9, 0.99, 0.999) and children number ($n$=2, 10, 50, 200, 500), this table shows the maximum number of faults $k$ with a probability $c$ equal to 99, 95 and 65 %.

For instance, considering the illustrative application of Figure 3.11a, the parent task $\tau_1$ has 2 children $\tau_3$ and $\tau_4$ to map (i.e. at least 2 tries). If each node has a reliability probability of 0.9, following Table 3.2, it is guaranteed at 99% that no more than 2 faulty nodes will be hit. Consequently, with at most 4 tries, the 2 children will be mapped successfully.

According to the Table 3.2, the nearby mapping strategy guarantees that, with a realistic reliability ($P_s > 0.9$), and even with a larger number of children, the number of faulty nodes hit will be negligible, and thus the number of extra tries. Therefore, the application execution is guaranteed, even if some nodes failed. In effect, our technique is able to find efficiently available fault-free nodes to map tasks again, without stopping the application.

| $c$ | 99% | | | 95% | | | 65% | | |
|---|---|---|---|---|---|---|---|---|---|
| $P_s$ | 0.9 | 0.99 | 0.999 | 0.9 | 0.99 | 0.999 | 0.9 | 0 .99 | 0.999 |
| $n$=2 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $n$=10 | 4 | 1 | 0 | 3 | 1 | 0 | 1 | 0 | 0 |
| $n$=50 | 11 | 2 | 0 | 10 | 2 | 0 | 7 | 1 | 0 |
| $n$=200 | 34 | 6 | 1 | 30 | 4 | 0 | 26 | 3 | 0 |
| $n$=500 | 75 | 11 | 2 | 68 | 9 | 2 | 62 | 7 | 1 |

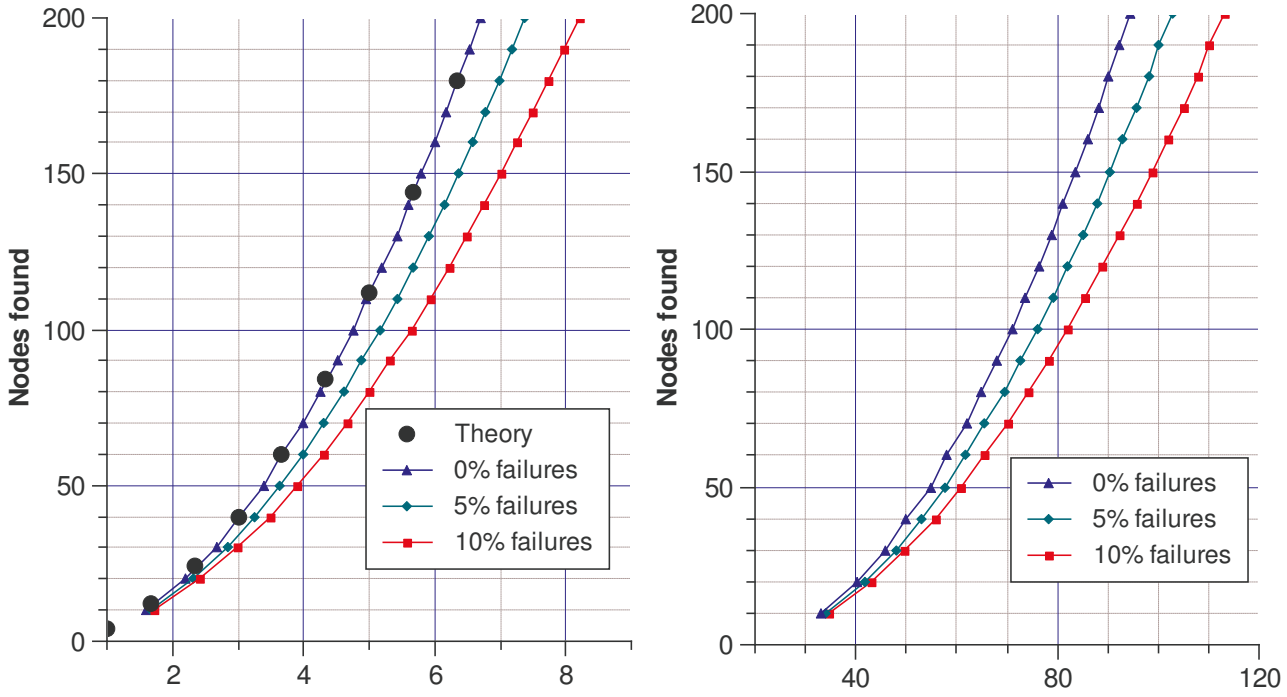
Table 3.2: Maximal number of defective nodes hit with a probability $c$ with $n$ children and a node reliability probability $P_s$

### 3.5.5 Validation

In order to validate the theoretical results given in the previous section, simulations were achieved based on the simulator and routing algorithms introduced in Chapter 2. For the validation of the proposed mapping strategy, the Variant A algorithm presented in Section 2.4.2 is utilized.

The simulation of faulted 2D-Meshes of $32 \times 32$ nodes is executed on a cycle accurate simulator, which enables accurate simulations. Faults were set randomly amongst the mesh nodes set (e.g. 102 nodes for 10% of node faults), while avoiding mesh partitioning. During simulation, the mesh's center node used Nearby mappin strategy for the mapping of 200 nodes ($n = 200$). It emits sequentially a

request to each of its neighbors using the Variant A routing algorithm; destination nodes being ranked in increasing Manhattan distance from the center node. The simulation stops when the center node received 200 positive acknowledgments. Figure 3.13 shows average results over more than 60 fault patterns for each fault rate. For all fault patterns, the algorithm succeeded in finding 200 fault-free nodes, with respect to Table 3.2.




(a) Average number of hops

(b) Average latency in *ns*

Figure 3.13: Impact of node faults on proposed mapping strategy using the Variant A routing algorithm

In Figure 3.13a, the simulation results are shown for the average number of hops for reached nodes for different $n$ and node fault rate $(1 - P_s)$, according to Section 3.5.4. Additionally, the set of points $\left(\overline{\text{Hops}}(H), \text{Nodes}(H)\right)$ defined in Equations 3.13, 3.12 are shown as black circles, and fits perfectly to fault-free case simulation results (blue curve). Figure 3.13b shows the average two-way latency for the reached nodes during simulations, and is similar to Figure 3.13a, since the network is not congested. For $n = 200$, the average latency is 94.4, 103 and 113 [ns] and the average number of hops is 6.7, 7.4 and 8.2, respectively for fault-free case, 5% and 10% of router faults.

Therefore, the performance degradation of our solution is reasonable, even with the simple Variant A routing algorithm. However, higher fault tolerance can be attained with more complex routing algorithms, when necessary. Thus, these simulations show how our technique, which does not require any knowledge of the system, tackles the uncertainties on interconnect and application levels. Our solution is more effective than broadcast-based techniques, which require to collect, store and maintain information about the whole neighborhood of a parent task, which may be prohibitive for large scale interconnection networks. In contrast, our proposition simply proceeds to several mapping steps. In the ever-changing context of many-core processors, our technique provides good adaptability of the application to the chip status, as shown in Figure 3.13, while requiring few traffic and processing power.

In the Section 3.5.2, a mapping technique oblivious to the application tolerating versatile fault patterns was presented. The proposed algorithm is capable of mapping tasks on an arbitrary number of suitable nodes $n$ (i.e. reachable and satisfying task requirements), with no preexisting knowledge on the network state, as shown in Figure 3.11. Moreover, our technique supports the re-mapping of tasks which

node has failed, as shown in Figure 3.12. Finally, in Section 3.5.5, a routing algorithm was presented, that could implement the proposed algorithm very efficiently, based on the exhibited simulation results.

### 3.5.6 Case study: the Motion JPEG 2000 application

In order to show its feasibility, our technique was analyzed in the case of the MJPEG 2000 decoder [125], a real application with a high potential for parallelism. Before to briefly explain the MJPEG 2000 application and the details results, let us begin by describing the estimation methodology.

#### Estimation Methodology

The messages duration exchanged between a parent task and its children are composed of four parts. The first part represent the average value per node for searching an available fault-free node, which is given in Figure 3.13b. For example, to find 100 fault-free nodes, the average latency per node is 82ns at 10% of fault. The second and third parts are the duration of the program code and input data transfer. The fourth one represents the duration for transfering the obtained result from the children to its parent task. In the following, the *Initialization* phase ($Init$) is equal to the sum of the first three parts and the *Result* phase ($Res$) presents the last one. The *Initialization* ($\Delta_{Init}$) and *Result* ($\Delta_{Res}$) phases duration are presented in the Equation 3.14 and 3.15.

$$\Delta_{Init} = \Delta_{search} + \Delta_{flit} * (codeSz + dataInSz) \quad (3.14)$$

$$\Delta_{Res} = \Delta_{flit} * dataOutSz \quad (3.15)$$

Where $\Delta_{flit}$ is the average flit transfert duration, $\Delta_{search}$ is the average latency to find an available fault-free node and $codeSz$, $dataInSz$ and $dataOutSz$ are respectively the program code size, the size of the data to be processed and the result size, all in flits The Computation-to-Communication Ratio (CCR) is derived from Equation 3.14 and 3.15 and presented in Equation 3.16, where $CT$ is the child Computation Time.

$$CCR = \frac{CT}{\Delta_{Init} + \Delta_{Res}} \quad (3.16)$$

Thus, the *Execution Time Reduction Ratio* ($RR$) is given by the *Sequential Execution Time* divided by the *Total Execution Time*, which is the sum of one child computation time ($CT$) with the *Initialization* ($\Delta_{Init}$) and *Result* ($\Delta_{Res}$) phases duration. The *Sequential Execution Time* can be approximated by multiplying the number of children by the computation time ($CT$) of one child, where $n$ is the number of children.

$$RR = \frac{n.CT}{CT + \Delta_{Init} + \Delta_{Res}} \quad (3.17)$$

#### Motion JPEG 2000 overview

From the MJPEG 2000 encoder side [125], the input data are partitioned in several ways. Every frames are independent and their components (i.e. Y,U,V which are also independent) can be decomposed

into a regular grid of tiles. Each tile is then split into sub-bands by the IDWT[21], depending on the number of decomposition levels. Sub-bands are further organized into roughly regular blocks called code-blocks, processed independently by the tier-1 encoder. Then, tier-2 encoder packetizes the data into streams. The number of tiles ($numXtiles$, $numYtiles$) is calculated based on the user-defined tile size, respectively $XTsiz$ and $YTsiz$ in X and Y direction. For each tile, there are $N_r$ different resolutions level($r$), results in several sub-bands. Consequently, given the code-block sizes $x_{cb}$ and $y_{cb}$ and the tile size $XTsiz$ and $YTsiz$, respectively in X and Y direction, the corresponding number of code-blocks $numXcblk$ and $numYcblk$ inside a tile can be derived. The code-block size is restricted by the standard and must belong to the interval $[2^2, 2^6]$ in each direction. Thus, the maximum number of code-blocks is obtained when $x_{cb}$ and $y_{cb}$ are equal to $2^2$. Let us consider a CIF frame of the size 352x288 with only one component (i.e. luminance) and only one tile. Let consider the smaller code-blocks size with only 16 samples each (i.e. $x_{cb} = y_{cb} = 2^2$). Then, the number of code-blocks per frame is $\frac{352}{2^2} \times \frac{288}{2^2} = 6336$. In this last case, code-blocks can be viewed as "fine grain" (only 16 samples each). This example shows the great potential for parallelism of the MJPEG 2000. This data parallelism fits well with the master-slave DAG model, and therefore the proposed self-adaptive mapping approach.

The MJPEG 2000 decoder is composed of three main sequential functional blocks. At first, the "un-packetization" function is done in so-called tier-2 decoder. Then, in tier-1 decoder, the arithmetic decoding is processed, followed by the IDWT, which generates the decoded tile. The standard [125] defines how data are processed and organized into streams. This document exhibits the great potential for parallelization of the MJPEG2000 decoder. Figure 3.14 shows a detailed fork-join DAG representation of one tile decoding part. All terms used in the following can be found in the standard ([125]). *Generally, the tier-1 decoder stage of the MJPEG 2000 decoder is consuming most of the time. Therefore, tier-1 focuses our analysis of the MJPEG2000 decoder's parallelisation, which results are shown in Table 3.3.*

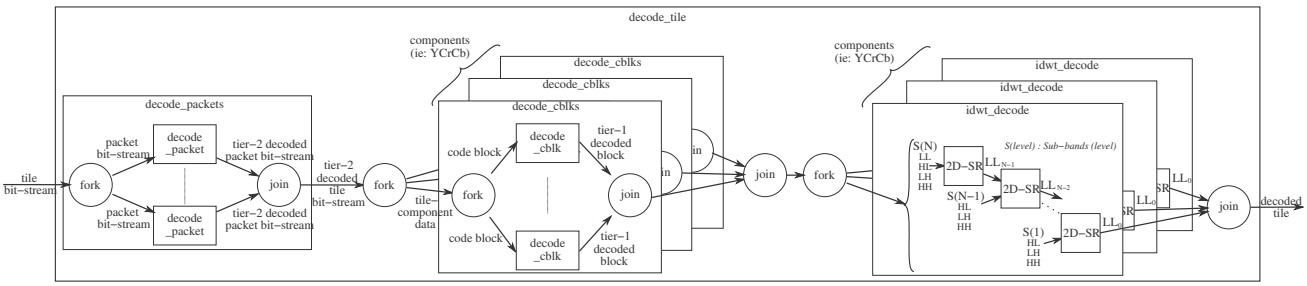



Figure 3.14: DAG of the MJPEG 2000 decoder

The duration of the Tier-1 decoding, which consumes most of the computation time, was estimated experimentally. A workstation with an Intel 2.4GHz processor and 3GByte of memory was utilized for these experiments, for a "waterfall" CIF movie sequence at 30 frames per second. The Table 3.3 depicts the average computation time by code-block in column 3. The column 1 gives the number of samples per code-block (i.e. code-block size), ranging from 16 (4x4) to 4096 (64x64). Column 2 is the number of code-blocks per frame. For example, with a code-block size of 32x32, there are 99 code-blocks per frame. Therefore, up to 99 tasks can be launched concurrently, each one having 1024 samples to compute (e.g. 4KB in the integer case).

The estimation methodology presented above allows us to estimate the different key parameters for the Tier-1 decoder. One key parameter is the *Communication time* derived from Equations 3.14, 3.15, and is shown in Table 3.3 columns 4 to 6 at 0, 5 and 10% of faults. From the tier-1 stage implementation, the program code size has been estimated around 4kflits without any compiler optimisation. The data to be treated are unknown a-priori, because they depend on the encoder parameters and the original movie.

[21] Inverse Discrete Wavelet Transform

Since the considered application is a movie compression/decompression application, a compression ratio at least equal to $\frac{3}{2}$ is considered. Then, the data size to be decompressed is at least $\frac{2}{3}$ smaller than the original data size (noted $Sample$ in Table 3.3 column 1). Consequently, the total transfer duration (Communication time) between a parent and all its children tasks is equal to $\Delta_{comm} = \Delta_{Init} + \Delta_{Res}$. Equations 3.14, 3.15 lead to $\Delta_{comm} = \Delta_{search} + \Delta_{flit} * (codeSz + \frac{2}{3} * sample + sample)$, where $sample$ is the number of samples per code-block, as depicted in Table 3.3 column 1. The program code size ($codeSz$) equals 4k flits and the time to find an available fault-free node ($\Delta_{search}$) is given by Figure 3.13b. The flit duration ($\Delta_{flit}$) is selected to be 1 nanosecond. The computation-to-communication ratio ($CCR$) is estimated from Equation 3.16 and depicted in Table 3.3, columns 7 to 9.

Table 3.3: Tier-1 Computation and Communication time estimation at processing frequency of 2.4GHz

| Sample / cblk | cblk / frame | Comp. time / cblk ($\mu s$) | Comm. time($\mu s$) ($\Delta_{comm}$) at % of failure | | | Comp. to Comm. ratio ($CCR$) at % of failure | | |
|---|---|---|---|---|---|---|---|---|
| | | | 0% | 5% | 10% | 0% | 5% | 10% |
| 16 | 6336 | 5.06 | 5,04 | 5,23 | 5,65 | 1 | 0,967 | 0,896 |
| 64 | 1584 | 20.2 | 4,45 | 4,52 | 4,62 | 4,54 | 4,47 | 4,37 |
| 256 | 396 | 76.75 | 4,48 | 4,5 | 4,51 | 17,1 | 17,1 | 17 |
| 1024 | 99 | 253.99 | 5,19 | 5,19 | 5,2 | 48,9 | 48,9 | 48,8 |
| 4096 | 24 | 744.24 | 8,23 | 8,24 | 8,24 | 90,4 | 90,4 | 90,3 |

According to the columns 4 to 6 of Table 3.3, the communication time is insignificantly impacted by the time to find an available fault-free node ($\Delta_{search}$). It is a small portion of the communication time, even at 10% of faults. Furthermore, the $CCR$ is higher for code-block sizes larger than 256, ranging from 17 to 90. In addition, the obtained *Execution Time Reduction Ratio* ($RR$) (derived from Equation 3.17) is close to the ideal one ( i.e close to $n$). For example, given a code-block size of 4096 (Table 3.3- last line), the communication time at 10% of faults (column 6) is $8\mu s$ and the computation time (column 3) is $744\mu s$. The sequential time is calculated by multiplying the number of code-blocks by the computation time of one code-block ($24 \times 744\mu s$) and is equal to $18ms$. Then, assuming that all messages can be sent or received in parallel, the $RR$ is 24, which is very close to the ideal case.

## 3.6 Variation-aware task mapping for application energy efficiency

In this chapter, the second main contribution is the improvement of the fault-tolerant mapping technique presented in Section 3.5 in order to improve the performance of applications running on unreliable many-core processor chips. Two generic task mapping strategies are proposed to improve the energy efficiency of these applications, and compared for a synthetic application. The nearest neighbor strategy is used as a baseline, and minimizes the communication overhead. Then, a novel energy criterion is introduced to balance the computation and communication energy consumption. While increasing the communication energy, this strategy reduces the overall consumption by up to 20%. Finally, a mapping strategy based on variability regions improves slightly the energy efficiency of the application in the presence of systematic variations.

### 3.6.1 Root task mapping

The selection of the node $n_{root}$ executing the root task has a significant impact regarding the application performance. In effect, the lack of available processing nodes within short distance of the root task will incur longer message routes and therefore increased application energy and latency.

#### Interconnect centering

Under random-dominated variability scenario or low variability rates, the optimal position for the root task is close to the center of the processor, following Equation 3.18. In effect, the interconnect center minimizes the average distance to other nodes. However, in the presence of multiple applications, center nodes may rapidly become congested. As a consequence, a more thorough algorithm such as region centering is often required.

$$n_{root} = \mathcal{C}\left[\frac{W}{2}, \frac{W}{2}\right] \tag{3.18}$$

#### Region centering

The Region centering adapts to the presence of regions, which appear in the presence of strong systematic variations and/or preexisting variations.

Discovering the central node of the best region for a particular application $A$ is generally-speaking a complex problem, therefore several heuristics are required. Firstly, variability regions are usually convex. Thus, the symbol $\square(n, app)$ is introduced as the square region centered on $n$, of width $\sqrt{card(A \triangleright T)}$, where $card(A \triangleright T)$ is the number of tasks of the application $A$. Second, a single value should reflect the efficiency of each core. In this work, the maximum frequency $n \triangleright f_{hi}$ of each node $n$ is chosen (higher is better). A different metric may be prefered to reflect the system requirements and specificities. The *region fitness* criterion is given in Equation 3.19, as the sum of the computing slack of nodes within the square region centered on $n_c$.

$$n_{root} = \underset{n_c \in \mathcal{C}}{\operatorname{argmax}} \left( \sum_{n \in \mathcal{C} \cap \square(n_c, A)} \left( n \triangleright f_{hi} - \sum_{\tau \in \mathrm{Mapped}(n)} \tau \triangleright \mathrm{Req} \triangleright w \right) \right) \tag{3.19}$$

### 3.6.2 Children tasks mapping

The mapping of children tasks obeys to other constraints than root positioning. In effect, the efficiency of a child task itself is considered, and includes its computations and the communications towards its parent task. Hence, several approaches are devised to map children tasks efficiently.

#### Nearest neighbor

Regarding the communication overhead, the optimal task mapping strategy is to map dependent tasks as close as possible. Therefore, the Nearest neighbor strategy has emerged. Child tasks are mapped to the suitable node which is the closest from the node that runs their parent task $\tau_p$. The trivial associated costs presented in Equations 3.20, 3.21 offer a lower complexity for the mapping algorithm.

$$\text{NearestCost}(n_{Pa}, n, \mathsf{Req}) = 0 \tag{3.20}$$

$$\text{NearestCost}_{min}(n_{Pa}, n, \mathsf{Req}) = \infty \tag{3.21}$$

#### Minimum energy

The variability has a significant impact on the cores performance, and the task mapping should consider this parameter. Therefore, a variation-aware task-mapping technique is proposed to leverage the performance of the most efficient cores.

The variability has a strong impact on the core energy consumption. Indeed, in order to maintain the same performance level, higher average voltage is required, and the energy per operation increases according to Section 3.3.3.

In the Minimum Energy mapping strategy, the energy efficiency of the cores is tested along with the distance from the node executing the parent task. The mapping criteria, given in Equation 3.22, is derived from the energy expressions presented in Equation 3.5 and Eq 3.6. The aim is to balance the performance of the chosen core and the communication overhead, in regard to the specificities of the task workload and the amount of communication towards the predecessor. The proposed approach is also compatible with more complex energy models (e.g. including leakage, temperature and contention effects).

$$\begin{aligned}\text{EnergyCost}(n_{Pa}, n, \mathsf{Req}) =& \text{Req} \triangleright w \times E_{op}(n, n \triangleright f + \text{Req} \triangleright w) \\ &+ \text{Req} \triangleright \text{c}_{\text{Pa}} \times E_{flit}(n_{Pa}, n)\end{aligned} \tag{3.22}$$

#### Adaptive ending criteria

During the mapping stage, in order to minimize the energy, the processing nodes are visited one by one. In order to avoid visiting every nodes in the chip, an adaptive ending criteria is defined in Equation 3.23.

As explained in Section 3.3.3, the distribution of processing nodes' characteristics may be approximated by a normal law. Consequently, the processing node energy $E_{op}(n, f_{hi})$ defined in Equation 3.5

follows a normal law $\mathcal{N}_E(\overline{E}, \sigma_E^2)$, where parameter $\overline{E}$ and $\sigma_E^2$ are respectively the mean and the variance of the energy per cycle for the higher frequency, amongst processor nodes. Based on the observation that $P\left[E_{op}\left(n, f_{hi}\right) > \overline{E} - 3\sigma_E\right] \approx 99.73\%$, a lower bound of the task energy is given in Equation 3.23.

$$\begin{aligned} \text{EnergyCost}_{min}(n_{Pa}, n, \mathsf{Req}) = \quad & \mathrm{Req} \triangleright w \times \left(\overline{E} - 3\zeta\sigma_E\right) \\ & + \mathrm{Req} \triangleright \mathrm{c_{Pa}} \times E_{flit}(n_{Pa}, n) \end{aligned} \tag{3.23}$$

Based on the minimum bound of the task energy, the node search described in Algorithm 5 stops when the minimum possible energy cost of the following node to visit is higher than the best node found already. The $\zeta$ parameter is called *relaxation factor*, and permits to stop mapping before ensuring that there is no better node. This parameter varies in $[0, 1]$, where $0$ leads to a behaviour similar to Nearest neighbor, while $\zeta = 1$ requires to search processing nodes until there is no possible further improvement.

### 3.6.3 Experimental results

In order to assess the efficiency of the proposed variability-aware mapping techniques, a synthetic application was mapped on the processor model presented in Figure 3.4 under the various variability scenarios presented in Table 3.1. The synthetic application is composed of 200 tasks totalling 1000000 operations, and overall communications of 1000000 flits. The operating voltages $V_{dd,lo}$ and $V_{dd,hi}$ are respectively set to 0.6 [V] and 1.2 [V]. The average threshold voltage $\overline{V_{th}}$ is set to 0.3 [V]. Four mapping strategies are compared, following Table 3.4.

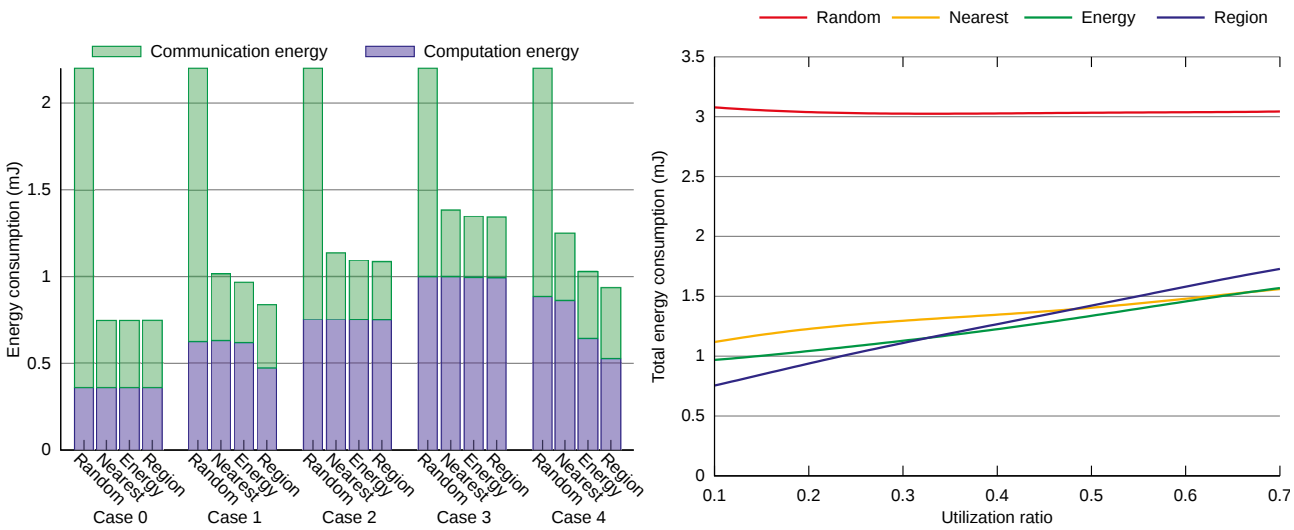
| Strategy | Root task mapping | Children task mapping |
|---|---|---|
| Random | Random | Random |
| Nearest | Interconnect centering | Nearest neighbor |
| Energy | Interconnect centering | Minimum energy |
| Region | Region centering | Minimum energy |

Table 3.4: Experimental mapping strategies

In Figure 3.15, the application energy is given under each of the variability scenarios defined in Table 3.1. The computation power and the communication power are shown for each mapping strategy in Figure 3.15a. In the variability scenario 0, where there is no variability, only the communication power is subject to improvement. In this context, there is no difference between the Nearest neighbor, Minimum Energy and Region. In other scenarios, systematic and random variations introduce difference in efficiency amongst the processor cores. In this context, the Minimum Energy outperforms the Nearest Neighbor strategy, by up to 20%. However, this technique tends to increase the communication energy budget. In the scenarios 1 and 4, variability regions appear due to the systematic variations. In these cases, the Region centering performs better than Interconnect centering. Overall, the gain is very limited, and may not justify alone the added computation complexity. However, in the presence of multiple applications, Region centering would improve largely the observed performance, by spreading applicaiton across the processor's nodes.

The utilization rate of the processor (i.e. the ratio of required processor's nodes) also has a substantial impact on the mapping strategy performance, as shown in Figure 3.15b. Under low processor usage, the Region mapping outperforms other strategies by leveraging the large amount of available nodes.

However, when the utilization rate is high, this mapping strategy places the root task too far away from the center, and there are not enough nodes for mapping the application efficiently. In this respect, the Minimum Energy strategy performs better by positionig the root task near the processor center.



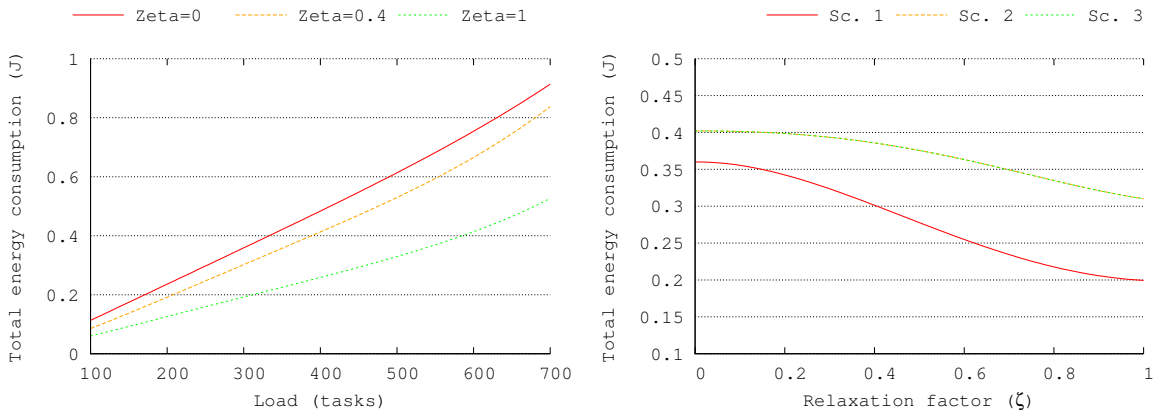

(a) Application energy consumption

(b) Utilization ratio under variability Case 4

Figure 3.15: Application energy consumption under various configurations

In Figure 3.16a, the total application energy consumption is given for different number of tasks and relaxation factors. Under variability Case 4 without faults, the application energy is reduced significantly by using the Minimum energy strategy ($\zeta = 0.4, 1$) instead that pure Nearest neighbor ($\zeta = 0$). Moreover, the relaxation factor $\zeta$ provides a continuous tradeoff between the application energy efficiency and the mapping cost in terms of communication overhead and applicative delay.

Fig. 3.16b shows the relation between energy consumption, variability and relaxation factor. Globally, when the variability increases, a higher relaxation factor is necessary for the application efficiency. This trend is observed as well when more faults are injected. In fact, when the heterogeneity between processing nodes increases, the most suitable nodes to use may be further, and more available nodes may have to bee visited, before these node are found.




(a) Task load

(b) Variability scenario

Figure 3.16: Application energy depending the relaxation factor

## 3.7 Conclusion

In this chapter, a self-recovering strategy was detailed and supported by different simulations and theoretical analysis. First, the application was decomposed as a DAG[22] of tasks, which guarantees the resilience of our approach in the presence of faults. Followingly, the Nearby mapping strategy maps inter-dependent tasks close to each other in Section 3.5.2. In Section 3.5.6, the proposed strategy is applied to the MJPEG 2000 decoder. In Table 3.3, the efficiency of the approach is given in terms of Communication time and Computation-to-Communication time ratio. It has been demonstrated that the proposed strategy allows MJPEG2000 decoder software to be parallelized and the execution time to be significantly reduced close to the ideal case, i.e. 24. Furthermore, it has been shown that the MJPEG2000 can be represented as a DAG of tasks, which allows to implement in an efficient way the proposed recovering strategy of parallel applications in the presence of faults in a many-core processor, consisting of up to 1000 computation cores.

This fault-tolerant approach is then improved to offer mitigate the impact of variability on future many-cores chips. A high level model was proposed, and several scenarios were depicted, depending on the evolution of the manufacturing process and circuit design trends. In the presence of high random and systematic variations, the Minimum Energy strategy reduces the energy consumption by up to 20%, in comparison to Nearest neighbor. In addition, when the systematic variations are important, Region centering has shown encouraging results, and the energy consumption is slightly reduced with regard to the Minimum Energy strategy. Finally an adaptive criteria for ending Minimum energy mapping more efficiently was presented.

[22]Directed Acyclic Graph

# Chapter 4

# VOCIS: A Versatile On-Chip Interconnect Simulation model for fault tolerance and performance assessment

In this chapter, a model named VOCIS[1] is presented. The principal objective of this software is to provide enough flexibility so that most fault-tolerant interconnects may be simulated under exactly the same conditions. Firstly, the model architecture is detailed in Section 4.2, with a focus on the design choices that enable genericity. Second, the exploitation of this model is depicted in Section 4.3. In particular, the distributed workflow is explained. Last but not least, several issues of choice regarding the analysis of interconnects behaviour in the presence of faults are discussed in Section 4.4.

## 4.1 Introduction

### 4.1.1 Motivations

The design space for unreliable many-core processors is extremely rich, due to the profusion of available techniques and stringent constraints. Hence, the comparison of heterogeneous solutions is often of interest. Unfortunately, when designs are too disparate, a comparative study often requires months of re-design, since the redaction low-level RTL[2] model of both approaches is required. In particular, many interconnect architectures have their own requirements, which are met by means of an *ad hoc* simulator, generally hindering the comparison against other advanced architectures.

The proposed model aims at providing a comprehensive base for comparing highly heterogeneous interconnect architectures, with various levels of abstraction. This unique versatility is especially vital for fault tolerance experiments, since interconnect fault tolerance is obtained by combining multiple complex mechanisms, each being subject to several design choices and calibration parameters. In particular, fault-tolerant routing algorithms often incur unique architectural choices, as discussed in Section 2.2.2.

[1] Versatile On-Chip Interconnect Simulation model
[2] Register Transfer Level

### 4.1.2 State of the art

Since multi-core processors based on NoC[3] received much attention from academia and companies, models for these objects have been developped by many contributors. Simulators are often split between *cores + cache* models (e.g. GEM5 [126]) and *interconnect* models. The sooner would simulate inter-core communications with little accuracy and the later would utilize synthetic or trace-based traffics (e.g. NOXIM [127]) for feeding the interconnect. Recently, the Omnet network simulator was extended to NoCs through the versatile HNOCS model [128]. Alternatively, *full-system* models are required in order to estimate the complete system behaviour. In effect, researchers have highlighted the impact of interconnect model on the accuracy of system simulations. In particular, the General Execution-driven Multiprocessor Simulator (GEMS) was combined with different interconnect models [129, 130]. The Simics simulator [131] also offers a commercial platform, which supports an extensive set of devices.

Another aspect of interconnects modeling is the chosen level of abstraction. Since high levels of abstraction require little computation power -at the expense of accuracy-, these models are relevant for *cores + cache* and *full system* simulations as long as the interconnect is utilized with a limited traffic (i.e. far from the saturation zone). These models are denoted as *transaction-level* models, since they process each transaction (i.e. each message or set of associated messages) at once, and emulate latency and interconnect load analytically. The *cycle-accurate* models actually simulate the transmission of messages within the interconnect, but of course require more simulation time. At this level of abstraction, most delays in the message transmission are modeled, such as crossbar arbitration and link propagation. Hence, the measurement of average latency is close to its actual value. [132] observes that *cycle-accurate* models generally provide a satisfying tradeoff between accuracy and simulation time, for timing analysis. Additionally, energy and area may be estimated by the Orion model [133], and temperature is often obtained through the HotSpot model [134]. Most models of this kind are written with the SystemC library, which provides primitives for the simulation of concurrent interacting modules. However, researchers raised concerns about the efficiency of this library for large scale NOCs [132], since its generic simulator core may utilize many OS[4] threads, and incur significant context swithcing overhead. At last, *cycle-accurate-bus-accurate* models also model the logical gates and buses, in order to obtain exact results. These models generally utilize the interconnect description at the RTL level, which also enables energy, silicon area and temperature measurements after synthesis.

To our knowledge, there does not exist simulators that initially target defective On-Chip interconnects. In effect, most publications on fault-tolerant architectures augment regular simulators, or directly develop an RTL model. However, this workflow renders the comparison of different techniques more difficult. In addition, few simulators are able to simulate generic routing algorithms over different interconnect topologies, which hinders the developpment of alternative interconnect architectures.

## 4.2 A generic model

In this section, details regarding the flexibility of the VOCIS model are given. The overall architecture is first described in Section 4.2.1. Then, the benefits of the generic graph representation are highlighted in Section 4.2.2. Followingly, Section 4.2.3 illustrates the capacity of VOCIS to handle versatile routing and arbitration algorithms. A third genericity issue (packet injection) is discussed in Section 4.2.4. Finally, we depict the mechanisms exploited to cope with the abundance of different configurations in Section 4.2.5.

[3]Network on Chip
[4]Operating System

### 4.2.1 Architecture

The VOCIS model has been written in C++, an object-oriented programming language based on C. The Boost library [135] was included for many core features of the software, and VTK [136] was utilized for the optionnal GUI[5]. The model is also compatible with the SystemC library, and system-level simulation exploits this compliance for simulating complex nodes behavior. The main development drivers were genericity regarding supported interconnect architectures, and reproductibility of the -somewhat complex- experiments.

In Figure 4.1, an overall scheme of the Vocis architecture is presented, following the UML notations. This figure highlights backbone classes (empty fill), which provide baseline functionality, and the main "factorized" classes (yellow fill), which are customized to suit the simulation targets requirements.

In this section, the **factorization** refers to the software mechanism utilized for the generic creation of objects. In this object-oriented design pattern, objects are created by a so-called factory method, which returns a reference to the newly allocated object. This pattern offers seamless genericity regarding created objects, since any object implementing the base functionality may be returned.

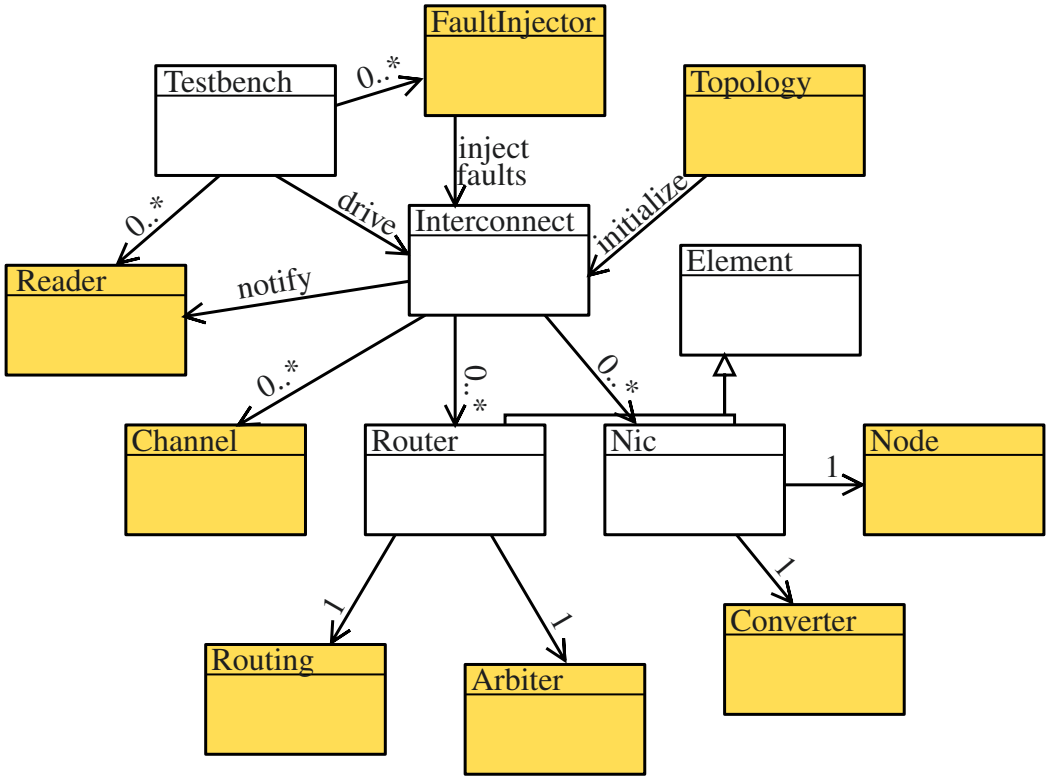



Figure 4.1: An overall view of the VOCIS architecture

The top model classes are named Testbench and Interconnect. The testbench is responsible for the proceedings of the simulation, while interconnect aggregates the simulated NoC resources (i.e. routers, NICs and channels) and provides methods for the access and operation of these objects. At simulation start-up, a Topology object is optionally created for initializing the Interconnect. A topology instance describes how routers and NICs are connected with channels, following the user-defined attributes (cf. Section 4.2.5). Then, the set of Reader objects associated to the Testbench are registered to the Interconnect. During simulation, these Reader objects will capture events from the Interconnect and store measurements based on these events.

[5]Graphical User Interface

In our model, the Router and NIC classes inherit from the abstract Element class. This relation simplifies routine operations and enables graph-based functionalities (e.g. route search). Each Router owns one Routing object responsible for the storage of router-level routing information and for the implementation of the routing algorithm at hands (cf. Section 4.2.3). Flit-level routing information are stored in a factorized FlitInformation object attached to Ack[6] and head flits, though actual storage of routing data within the flits content is also possible. An Arbiter object is also associated to each Router instance, and provides the crossbar and channel arbitration algorithms, as discussed in Section 4.2.3. Regarding NICs, there are also two customizable objects for each instance. First, the Converter transforms messages into packets and *vice versa* and manages ETE operations. Second, a Node object is responsible for the creation of messages and optionally the processing of received messages.

Finally, the interconnect's channels are described by the factorized Channel class. Each channel logically includes associated output buffers, link(s) and input buffers, following the taxonomy presented in Section 2.1.2. VCs[7] (introduced in Section 2.1.3) are supported natively. Since the Channel interface is generic, various channel designs may be represented regarding link type (e.g. asynchronous, shared FIFO, with repeaters, serialized) and flow control (e.g. Ack, credit-based, On/Off). Moreover, reduced models may be utilized as well, in order to reduce computation requirements.

Since this model was conceived with fault tolerance experiments in mind, a vector of customizable FaultInjector instances is appended to the Testbench, in order to inject faults in a generic way. During simulation, FaultInjector objects are triggered on each cycle, and insert faults into Element and Channel instances of the interconnect. The simulator handles multiple defect modes (e.g permanent, transient), and may be extended to support additional fault types.

### 4.2.2 Topologies

The development of NoC simulators is seasoned with several crucial architectural choices. The generality of the model regarding topologies is such a major decision. In effect, restricting to regular topologies such as 2D-Meshes and 2D-Torus (e.g. [127]) significantly reduces the code complexity and improves the simulation speed. Since the initial motivation for VOCIS was to support irregular topologies, a more generic approach was taken.

Following Figure 4.2, interconnects are represented by directed graphs. In this graph, vertices correspond interconnect elements, either NICs or routers. NICs are restricted to one output channel and one input channel, since they do not implement output channel selection (i.e. routing). Channels are represented by one arrow in each transfer direction. Hence, assymetric topologies are supported as well. Self-loops represent internal buffers that are required by some interconnect designs, such as the VS[8] described in Section 2.4.3. Finally, each vertex and arc is identified by an UID[9] and owns a defect mode attribute, that characterizes its current status.

Beyound the support of most existing topologies, the generic implementation of complex algorithms is another major advantage of representing interconnect as a graph. For instance, a generic shortest path algorithm from the Boost library was utilized for the emulation of an idealized routing, and for the detection of interconnect partitioning. In addition, the high-level simulation of several techniques requires the knowledge of close routers and channels. The graph representation naturally provides access to attributes of neighbor and $n$-hops away interconnect resources (i.e. routers, channels and NICs[10]).

[6] positive Acknowledgement
[7] Virtual Channels
[8] Virtual Source
[9] Unique IDentifier
[10] Network Interface Circuits

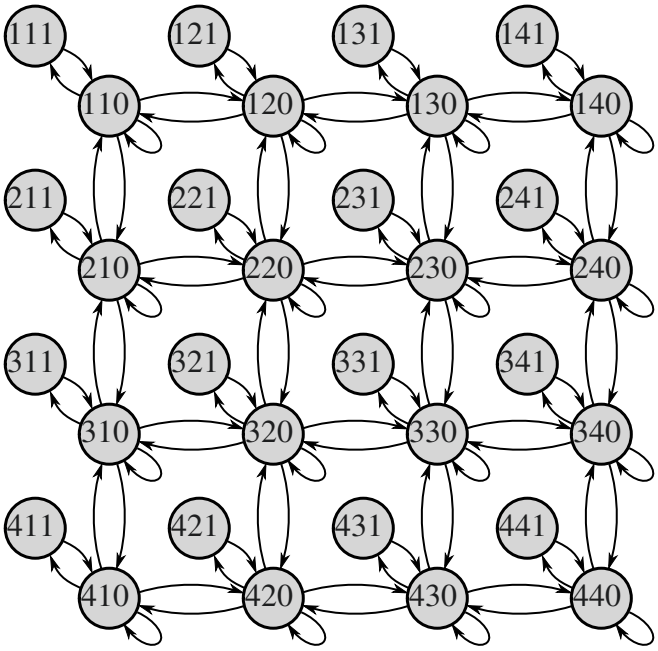



Figure 4.2: An example of interconnect graph

### 4.2.3 Routing and arbitration algorithms

Another distinctive feature of VOCIS[11] is the support for most routing and arbitration mechanisms of the litterature. This functionality is based on the graph representation of the router's channel dependencies presented in Figure 4.3. Followingly, adaptive and multicast routing algorithms are supported natively by VOCIS. The model also complies with most arbitration strategies and crossbar architectures.

In VOCIS, each router embeds a channel dependency graph similar to Figure 4.2, which stores the state of each transfer within the router. Each input (respectively output) channel -labeled chX- is connected to the **In** (respectively **Out**) group vertex. Similarly, each VC -labeled vX.Y- is tied to its enclosing channel. This hierarchical representation of the router's channels simplifies the programming of complex crossbar arbitration and adaptive routing algorithms.

Each arc from an input VC to an output VC corresponds to an ongoing packet transfer. The current state of the packet transfer is represented by the arc decoration. When the header of a packet reaches one input VC of the router, the routing function selects a set of **Candidate** output VCs to transfer the packet, alike VC v3.2 in Figure 4.3. On each cycle, the arbitration algorithm grants transfer of a flit to chosen input VCs. Whenever a Candidate transfer is awarded a grant, its state translates to **Candidate w/ grant** (e.g. v2.1) and, on the next cycle, to **Locked** transfer (e.g. v3.1). Again, whenever a Locked transfer receives a grant from the arbiter, its state temporarily changes to **Locked w/ grant**, alike v3.3.

The combination of the channel dependency graph presented in Figure 4.3 and the interconnect graph presented in Figure 4.2 offers substantial benefits. First, the detection of deadlocks is accomplished generically, by periodically computing cyclic channel dependency in the complete interconnect. While this functionality is merely usable for debug in deadlock-avoidance schemes, it provides precious insights for early stage deadlock-recovery techniques. This merger of channel dependency and interconnect graphs also provides a convenient data structure for tracking packets in wormhole routing, which is an important debug feature.

[11] Versatile On-Chip Interconnect Simulation model

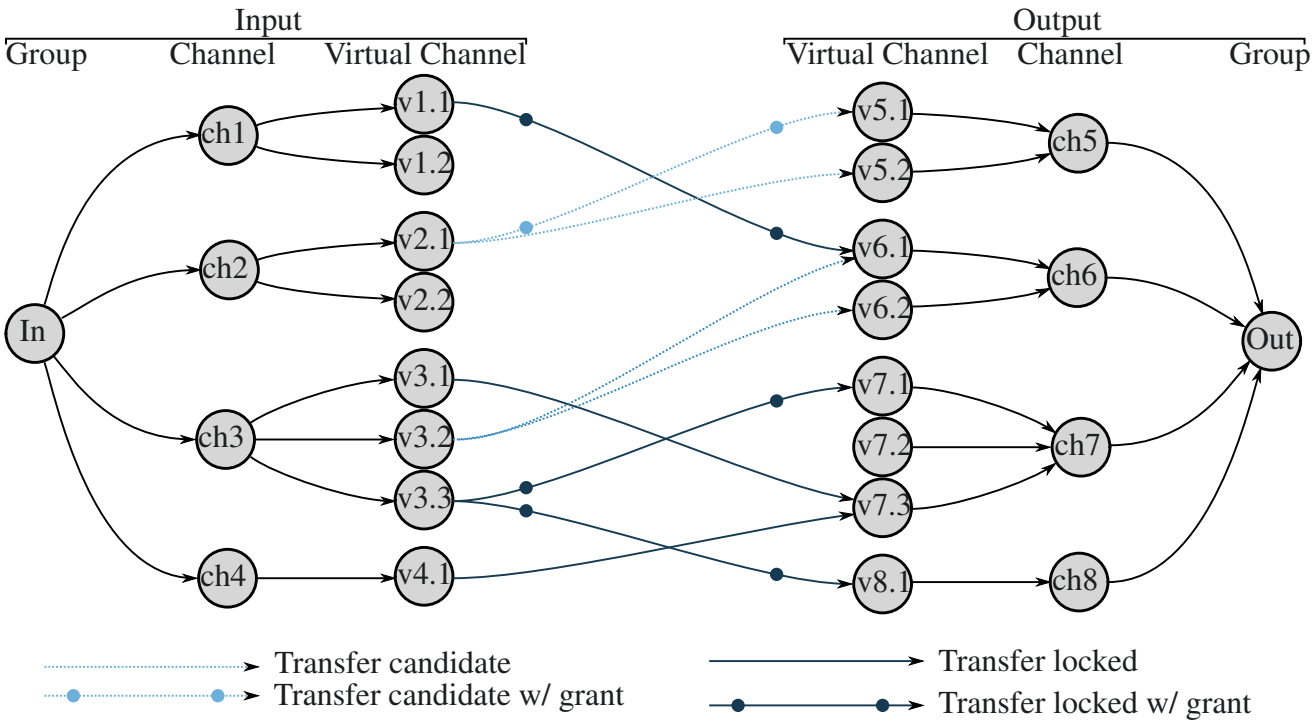



Figure 4.3: An example of router's channel dependency graph with transfers state

In the context of adaptive and/or multicast routing, the algorithm may request multiple output VCs for one input VC. A multicast routing algorithm will actually transfer the packet towards all output VCs, while adaptive routing will pick one among those granted by the crossbar arbiter. In these cases, the interaction of the routing and arbitration cannot be avoided. For the arbiter to cope with this situation, VOCIS provides two methods of the routing algorithm. The first detects whether a flit is actually blocked due to previous arbitration choices, i.e. the next flit cannot be transfered. A second method lets the routing algorithm selects the sufficient (and prefered) output VCs.

The arbitration process may also change significantly for one interconnect to another. Firstly, the connectivity between channels may deviate largely from complete crossbar. For instance, the Rotary Router proposed in [137] only connects channels in the clockwise order. Inversely, some channels may be connected by redundant buses such as crossbar bypass wires in [138]. In addition, the arbiter logic sometimes divert from the *VC + Crossbar* scheme described in Section 2.1.2. For instance, speculative arbitration of VCs improves the utilization rate of the crossbar. Finally, while most NoC arbiters are based on the Round Robbin policy, more complex algorithms may be implemented for improving performance under unbalanced traffic. In the case of hierarchical topologies for instance, weighted arbitration offers a substantial improvement [139].

### 4.2.4 Packet injection

The VOCIS model provides an interface that suits both synthetic and application-like traffic injection scenarios. For instance, the routing algorithms presented in Chapter 2 were simulated with synthetic traffic, while simulations for the fault-tolerant framework of Chapter 3 utilized application traffic. For application-driven traffic, VOCIS provides signals indicating whether a message was transmitted successfully. For synthetic traffic generation, VOCIS provides three essential parameters. First, the selection of destination nodes regroups the stochastic selection law and the rejection rule for invalid nodes. Second, the timing of messages emission generally follows one stochastic law for the emission of the initial message and another for the size of bursts. Finally, the NIC exhaustion policy determines

how the message emission requests above the NIC capacity are handled.

The selection of destination nodes has a large impact on the performance observed in the presence of defects. Firstly, the selection law impacts the observed fault tolerance and performance. For instance, utilizing a complete traffic (each source node sends a message to each destination node) provides worst-case results, both in terms of fault tolerance and performance. More commonly, the uniform selection of destinations offers a good overview of the interconnect characteristics, though it does intrinsecly balance the traffic. Alternatively, a local traffic leads to higher observed performance and fault tolerance, since this traffic only considers destination node within a selected range, which reduces the average number of hops.

The rejection of invalid destination nodes also affects the observed performance of the interconnect. In effect, a fault-oblivious selection law may choose faulty destination nodes. Since most interconnects give messages up only after one or more retransmissions occured, messages targeting defective nodes often obstruct NICs' storage and consequently the overall traffic is significantly reduced, since their service time is orders of magnitude larger. In practice, researchers either remove faulty nodes for the eligible destination nodes set or inhibit message retransmissions. In effect, real systems would memorize permanently faulty nodes and block traffic toward them. However, this assumption is biaised in the presence of intermittent faults and for large interconnects, since faults may occur too frequently for the system to keep track of all of them.

The emission of messages is usually triggered by an exponential law, which models memory-less events. Since applications often exhibit bursty traffic (i.e. several messages are sent shortly), a stochastic burst size is added to the injection mechanism. The Pareto law is chosen for determining the burst size, based on the litterature [140].

For injecting a synthetic traffic, there are two alternatives policies regarding NIC exhaustion. When the simulator requests a message injection that the NIC cannot process, either the message is dropped (i.e. rejected) or stored in a simulator queue (i.e. queued). In the latter scheme, the message injection time -used for latency computation- is set before message is stored in the simulator queue. This parameter completely redraws the outputs of the simulation and their meaning.

The message rejection models an injection process which depends on the interconnect state. In effect, if the interconnect is heavily loaded, the NIC is exhausted and message is not sent. From an applicative perspective, this corresponds to an application which tasks' execution mainly depends on the reception of messages from the interconnect. Hence, the emission of messages is slowed down as interconnect load increases. In the case of synthetic traffic simulations, the interconnect saturation is less distinct, but post-saturation analysis is available, since latency and throughput remain roughly stationary beyond the saturation point. However, the interconnect generally displays an oscillating behaviour, where internal state fluctuates largely along simulated time. As a consequence, latency plots require many simulations averaged together to obtain stabilized values.

Alternatively, message queueing has a major effect in the case of near-saturation traffic. In effect, messages stack up the nodes as the simulation advances, and for the larget injection rates, the message latency increases linearly with the simulated time. Graphically, latency plots show a clear asymptote near the saturation point, which enables a quantitative comparison between simulated configurations. From an application point of view, these conditions correspond to a streaming application running at constant applicative frequency and delivering periodically messages to the interconnect, oblivious to the interconnect's state. In this scheme, a criterion for the termination of simulations is required, to avoid exhausting the host memory and reduce the overall simulation time. In VOCIS, we propose the average criteria $\mathrm{Stop}$ presented in Equation 4.1, for a robust termination of simulations. $\mathrm{QueueSize}(n)$ represents the size of the node $n$'s queue, while $Q$ represents the nominal maximum queue size and $W$ the width of a 2D-Mesh interconnect. This criteria is robust to node faults or incidents (e.g. multiple

messages wait for timeout and block others), while limitating the latency range.

$$\text{Stop} : \sum_{n \in \mathcal{C}} \text{QueueSize}(n) < W \times W \times Q \tag{4.1}$$

In the general case, both approaches show benefits and limitations, and simpler readings contributed to push forward experiment utilizing message queueing. However, in practice, saturation points may vary largely from one configuration to another, thus finding the injection asymptote accurately may be tedious. Second, results may depend on the length of the simulation near saturation. Hence, in the context of fault tolerance for many cores chips, rejection of outstanding messages may be preferred. Besides, the application model is closer to the current trend for many-core chips; and running simulation on or beyond the actual saturation point is desirable when asserting the robustness of NoC[12] configurations.

### 4.2.5 Configurations profusion and simulation reproductibility

Since target systems are heterogeneous and fault tolerance research often incurs a try-and-compare approach, the configuration of simulators needs flexibility and readability. In addition, fine analysis of simulation results often requires to re-execute a simulation, to visualize some critical points. As a consequence, VOCIS includes a flexible configuration mechanism, based on objects configuration attributes. In addition, the generation of pseudo-random numbers is reproductible and composable.

Within the VOCIS model, most objects own a set of configuration attributes that pilot their behaviour. This approach offers fine-grain customization as well as configuration serialization suppport. In effect, the simulator is able to read the complete configuration from all configurable objects and save it to an XML file, based on the Boost library. Inversely, at start-up, the model reads several configuration files and command-line arguments to parametrize the model. The factory design pattern described in Section 4.2.1 is essential for this feature, to provide seamless configuration of the model, based only of a set of configuration attributes. In addition, objects may also feature configuration rules, to detect inconsistent configurations, alike a 3D-Mesh interconnect running a 2D-Mesh routing algorithm. The rule check is systematically run at start-up, after the initial configuration of interconnect objects.

Since single-threaded programs are completely deterministic by essence, the emulation of random events such as the occurence of faults, is based on so-called pseudo-random numbers. These are generated by specific algorithms, which output a sequence of numbers with no statistical relations. However, the output sequence of most procedures is completely determined by a single number named *seed*. In VOCIS, each object that requires random values embeds its own pseudo-random generator, and the generator seed is set by a configuration attribute. This methodology firstly ensures that the object will behave identically under different execution environments. Second, result analysis and debug may require to tweak a few configuration attributes before re-executing the simulation. In that case, VOCIS ensures that objects random number sequence will change only if its attributes were modified.

## 4.3 Experiment workflow

The simulation of complex NoCs subject to faults presents multiple challenges, especially regarding the production of representative results. At first, the comprehension of NoCs internal behaviour is improved by advanced model features for debug and visualization, as discussed in Section 4.3.1. Second,

[12]Network on Chip

the computation of proper experimental data requires large processing capacity. In this perspective, the distributed simulation methodology utilized by VOCIS is presented in Section 4.3.2. Finally, the processing of raw simulation data into readable figures requires a significant result analysis framework, that is discussed in Section 4.3.3.

### 4.3.1 Logging and visualization

The development of techniques for large defective interconnects often requires an in-depth knowledge of the interconnect behaviour. As a consequence, the logging features of the model improve the user experience, by providing instantly the more appropriate log outputs. In Vocis, the display of messages is centralized in a singleton logger object. Each message is identified by its source object's UID[13], the type of this source and the severity of the message. Then, the logger decides whether to output that message or drain it. The logger contains a list of allowed sources, each associated with a minimum severity, and a base severity is used for messages from all sources.

In addition to text information, VOCIS[14] provides a GUI[15] for the visualization of the interconnect behaviour. This optional feature is based on the VTK library for the interconnect interface component, and the KWWidgets library for GUI widgets. A 3D visualisation library was chosen in order to support avanced graphical operations (e.g. zooming, panning) and the support of opacity. These features allow for extension to 3D-Meshes for instance and the display of additionnal information along the $z$-axis such as node status or on-line statistics.

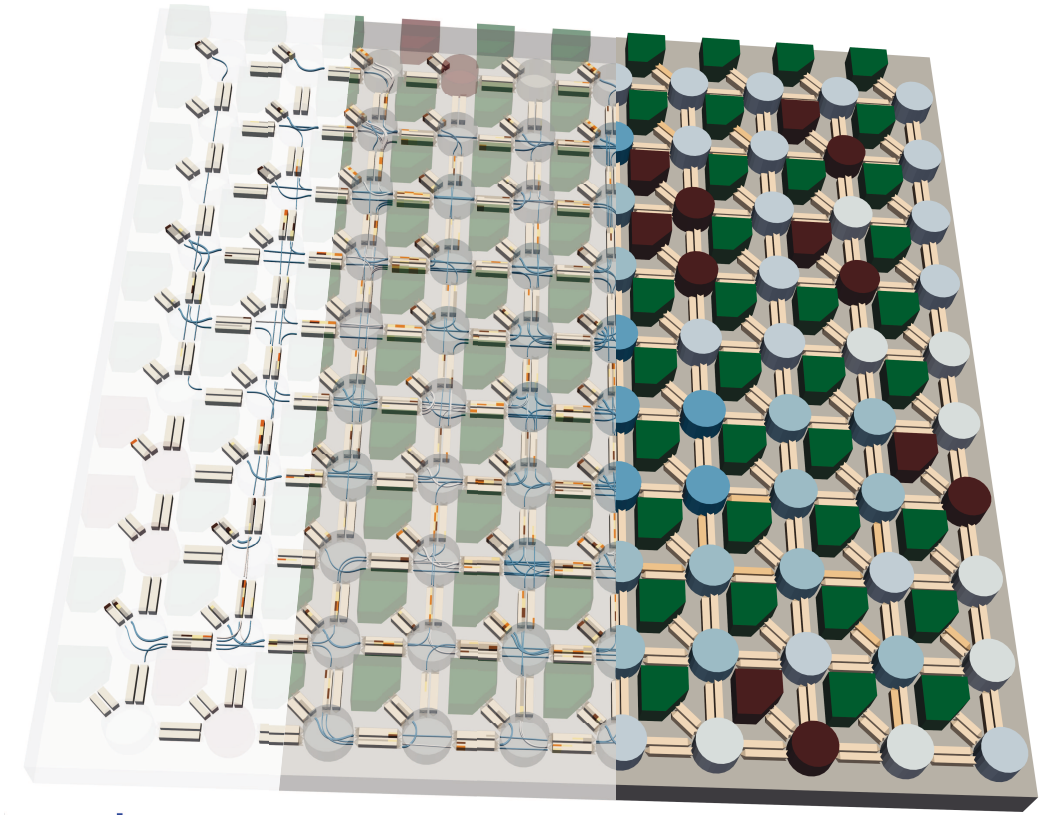


Figure 4.4: Snapshot of an interconnect simulation

In Figure 4.4, the three view modes for simulations capture are displayed. On the right-hand side, the contour mode shows the envelope of routers, channels and NICs. The color of each element reflects

[13] Unique IDentifier
[14] Versatile On-Chip Interconnect Simulation model
[15] Graphical User Interface

their internal state, in order to provide a global view of the interconnect state. On the opposite (left), content mode only displays in-transit flits and the routers internal state. This representation is rather complex but provides a detailed view of the interconnect status. In-between, the mixed mode shows both interconnect structure and internal state.

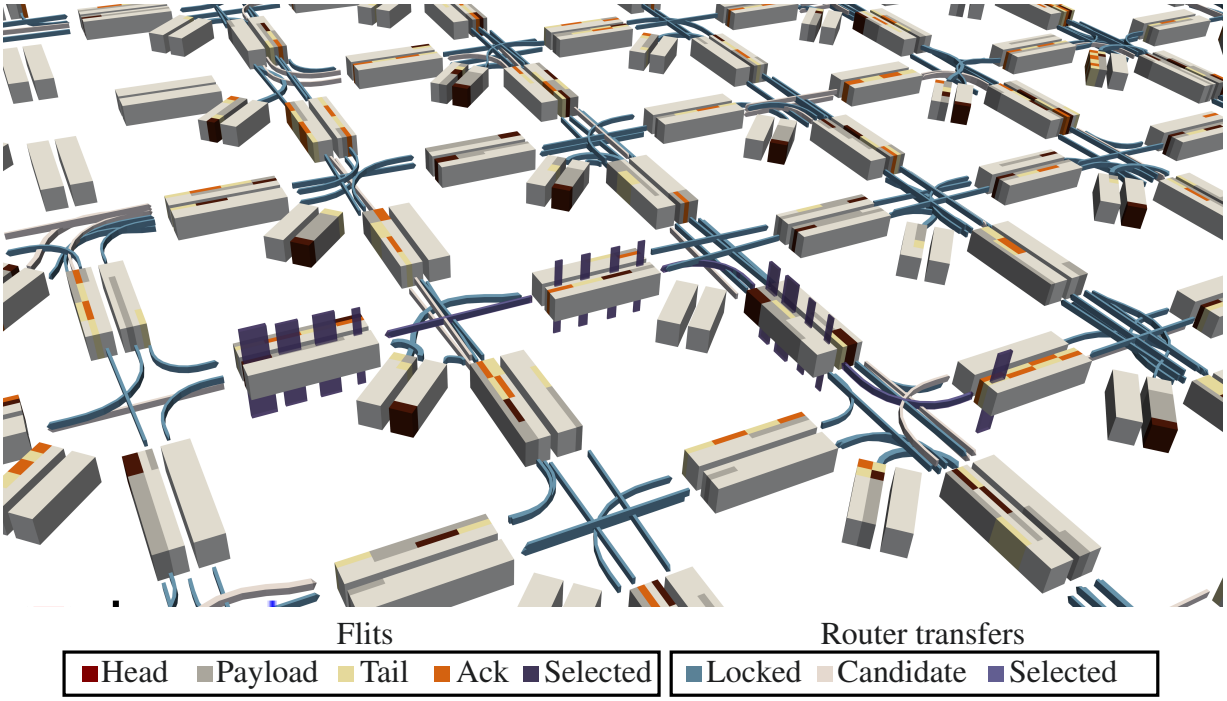



Figure 4.5: Detail of a content view

In Figure 4.5, an insight of the content mode view is shown. This picture shows the content of each channel, represented by a white rectangular solid. In this simulation, each channel is sliced into up to 4 VCs, and local channels include one VC towards NIC and 2 VCs towards router. The presence of a flit within channels is depicted by a colored surface on the channel's solid, following the figure legend. Moreover, the current status of router transfers is represented by colored arrows starting from the input VC to output VCs, as discussed in Section 4.2.3. We found that presenting both the content of channels and the status of routers transfer, improves the overall comprehension of the interconnect behaviour. Specifically, VOCIS proposes to track packets as they move throughout the interconnect.

The user may interact essentially in two ways. First, VOCIS allows a random access to computed states. As a consequence, the evolution of the interconnect may be displayed forward or backward, with different step speeds. In addition, the replay of simulations is available without requiring its re-execution. Second, user may change the camera position with the mouse, and more importantly, select any element of the 3D model. Then, the associated text information is displayed and the selected element is highlighed. For instance, in Figure 4.5, the head flit of a packet was selected, and all flits and ongoing router transfers are highlighted, so that the progression of the packet is readable.

### 4.3.2 Distributed simulations

A specificity of fault tolerance simulations is the size of the campaign's simulation set. In effect, multiple simulations are required for each configuration, since a minor modification of fault and/or message injection patterns may change significantly the simulation outcomes. As a result, the experimental results presented in this thesis required the computation power of multiple computers, and a distributed simulation methodology presented in Figure 4.6 was utilized.

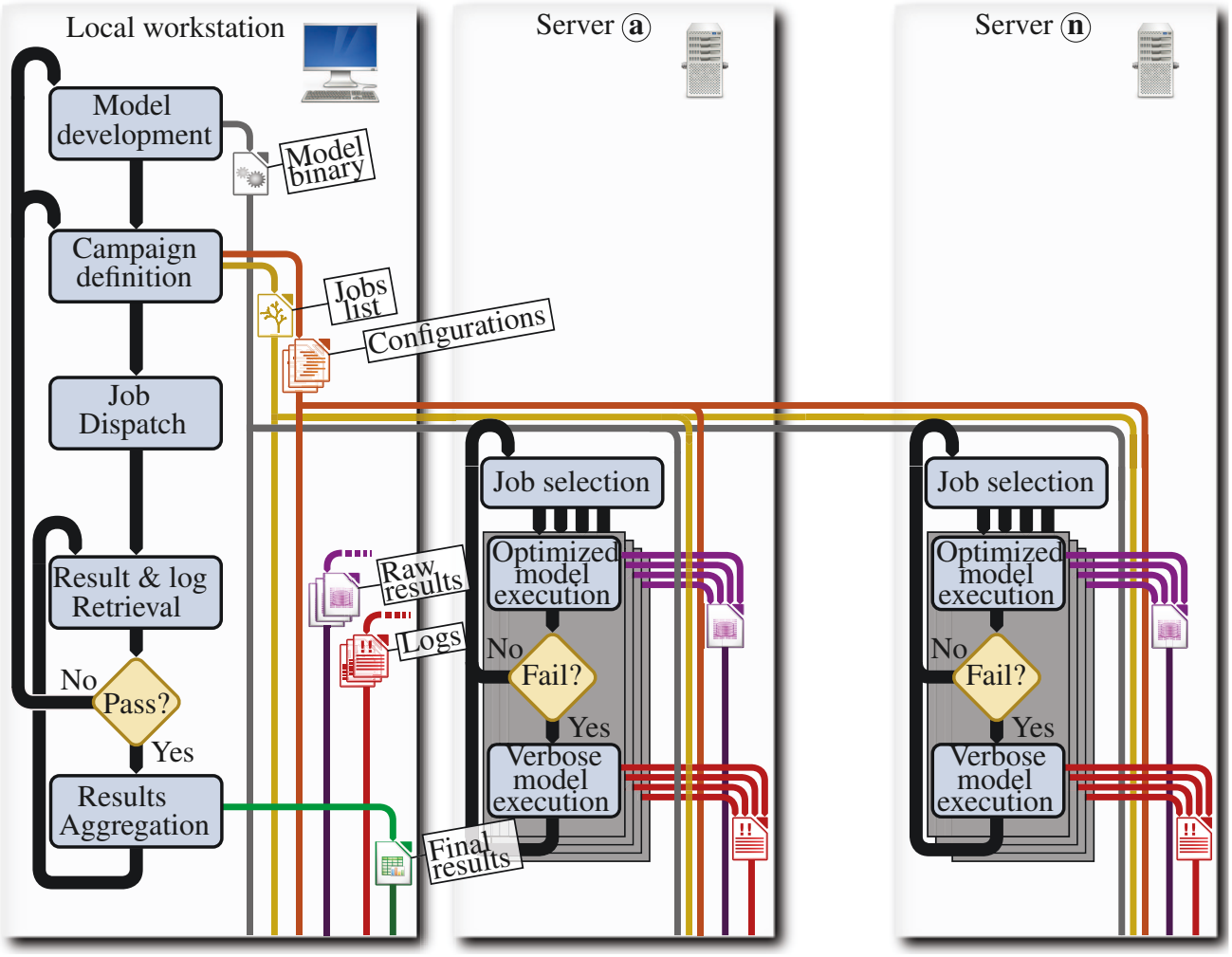



Figure 4.6: Work flow for distributed fault tolerance simulations

The parallelization of interconnect simulation is a complex task since each node is normally assumed to communicate with others with equivalent frequency. As a consequence, simulators witth a parallelized core incur a large synchronization overhead between the simulation threads. Instead, when simulations are independent and have limited computational needs, multiple independent threads may be utilized, without parallelization overhead.

The Figure 4.6 presents a simplified view of the distributed workflow utilized for VOCIS experiments. Once the model has been augmented with the desired features, the campaign is defined through a set of configuration files (c.f Section 4.2.5) and a list of simulation instances (a.k.a. jobs). Jobs are then dispatched among computation servers ⓐ to ⓝ.

Each server executes multiple jobs in parallel, in relation with the number of cores and the requirements of other users. The optimized (a.k.a. release) build of the binary is executed for each job. In this version, all the debug, verification and logging functionalities are disabled at compile time, in order to improve the simulation speed. However, if the simulation fails, the verbose (a.k.a. debug) build of the model is utilized, and output is saved in a log file. In the context where campaigns consists of thousands of jobs, this stage improves greatly the capacity to detect erroneous code (i.e. bug) or configuration (i.e. configuration flaw).

At the local workstation, a script periodically retrieves the simulation results and logs, either by downloading only novel files or a complete archive when there are many changes. Then, the user may choose to terminate the campaign when logs display a critical flaw in the campaign settings. Alternatively, the

user may consider that failed simulations are marginal and continue on with the current campaign. Lastly, since result files are stored at servers level, the re-execution of a campaign does not require the recomputation of all results. In effect, VOCIS skips simulations which result file already exists by default. Finally, the results are aggregated, based on raw result files from servers. At this point, a preliminary analysis (e.g. results ordering, inter-simulation averaging) is processed, according to Section 4.3.3.

### 4.3.3 Result analysis

A significant part of VOCIS was devoted to the implementation and comprehension of a post-processing framework that could handle the diversity and variety of the raw result data. In effect, the complex phenomena occuring inside large defective interconnects translate into heterogeneous measurements, alike average timing, event counting or timely evolution. These metrics require a flexible result storage and post-processing. Second, the obtained data are stained with noise and missing values. As a consequence, a data stabilization technique is required for obtaining representative stationary values, while non-linear regressions are utilized for characterizing variable values (e.g. linear, exponential).

#### A word about practical data

Since campaigns daily include thousands of simulation jobs, model configurations cannot be calibrated for measurements consistency across the complete field of exploration. For instance, lowest traffic rates and highest fault injection rates simulations lead to latency measures scattered with empty values. As a consequence, VOCIS handles these ”data holes“ not to distort the output of stabilization, regression or combination -e.g. summation, averaging or maximization-.

Second, since final results are based on the combination of multiple simulation jobs, the raw result files are required to provide adequate meta-data, such that final computation is not biased by this two-stage result computation. For instance, many measurements require the computation of average and standard deviation among sampled values. Hence, different measure modes are supported by VOCIS in order to reconcile storage requirements and available meta-data.

For illustration, the update of *complete* measures is presented with Algorithm 7. Since each measure is stored in a single vector -for memory and serialization reasons- the first vector element contains the samples duration. The data samples are ordered by simulated time blocks of identical period. Then, when a sample is to be added, the value of the adequate block is updated following Algorithm 7, creating empty blocks when required.

In order to preserve average and variance of each blocks, three real numbers are stored for each block. In order to modify the iteratively the average with additional samples, the block's average $avg$ and sample count $cnt$ are saved. Following Algorithm 7, the block residu $M_2$ is stored in the measure vector, to support the iterative computation of variance, based on the variance computation algorithm from Knuth [141]. . Following [141], the block variance $s$ may subsequently be obtained with Equation 4.2.

$$s = \sigma^2 = \frac{M_2}{cnt - 1} \tag{4.2}$$

**Algorithm 7** Iterative average and variance update

**Require:** $m$ the measure vector
**Require:** $val$ the value of the sample
**Require:** $T$ the time label of the sample

**procedure** UPDATEMEASURE($m$, $val$, $T$)
  $p \leftarrow m[0]$
  $i \leftarrow 3 \left\lfloor \frac{T}{p} \right\rfloor + 1$
  **while** $i + 1 \geq \text{size}(m)$ **do**
    $m \leftarrow m, \{0, 0, 0\}$
  **end while**
  $cnt \leftarrow m[i]$
  $cnt \leftarrow cnt + 1$
  $m[i] \leftarrow cnt$
  $avg \leftarrow m[i + 1]$
  $\Delta \leftarrow val - avg$
  $avg \leftarrow avg + \frac{\Delta}{cnt}$
  $m[i + 1] \leftarrow avg$
  $M_2 \leftarrow m[i + 2]$
  $M_2 \leftarrow M_2 + \Delta \times (val - avg)$
  $m[i + 2] \leftarrow M_2$
**end procedure**

### Data stabilization

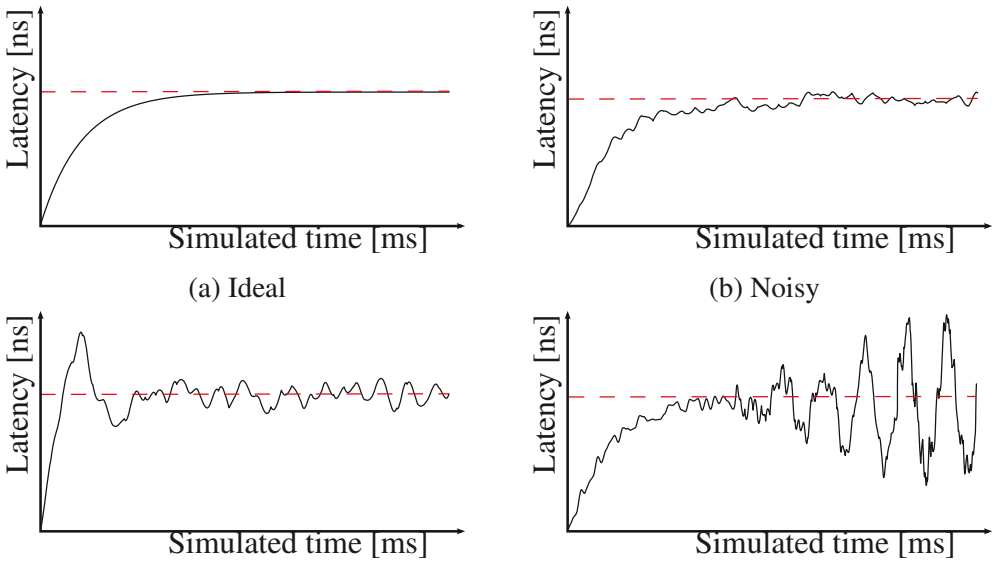
The simulation of faulty networks generally results in changing data, because of the interconnect instability. However, a single value representing whole data is required. Hence, an estimation scheme is required. Several approaches were considered during this thesis, and will be detailed in this section.

As a baseline, let us consider the ideal evolution of latency over simulated time in Figure 4.7a. Latency stabilizes *quickly*, and the permanent state measure is obtained by considering the last value in time. In addition, assessing whether the simulation was long enough for this parameter to stabilize is straightforward. In effect, the curve steepness decreases smoothly over time, and permanent state is reached without doubts when curve is -almost- flat.

Unfortunately, actual measurements are subject to significant levels of noise, such as shown in Figure 4.7b. There are essentially two approaches to that situation. Firstly, the sampling period may be increased in order to smooth the curve and return to the ideal case. Alternatively, a more robust estimation of the parameter may be used.

For instance, the **moving average** consists in the computation of the average over a continuous subset of samples. Then, the stability of the subset is measured either through standard deviation readings or maximum sample variation. This scheme also handles the presence of strong transient deviations (e.g. Figure 4.7c), since stability conditions are not met during the transient peak. When sliding the subset over data samples, the user hopes that some sample sets will meet the stability conditions. However, the sampling period may simply be too small for moving average to converge, alike Figure 4.7d. In that case, the sampling period may be increased artificially by averaging contiguous samples so that a novel moving average has more chances to converge.

In practice, while moderate simulation settings lead to parameter evolutions such as Figure 4.7b, the introduction of permanent faults and high traffic demands pushes the interconnect to a more oscillating




(a) Ideal (b) Noisy

(c) Transient oscillation (d) Permanent oscillations

behavior such as Figure 4.7d. In this case, moving average was not always sufficient, since it tends to reflect the oscillation noise. In this circumstance, the average of all samples may be used, yet the bias incurred by the parameter transient state will corrupt the obtained data. Alternatively, regression techniques may be exploited to reduce the sample set to an algebraic formula, and infer the permanent value robustly.

### Regressions

Data of interest may not always be stationary. For instance, the average latency across different injection rates changes rapidly around the saturation point. Unfortunately, these measures are perturbated by strong noises. As a consequence, regression of the resulting data is often required for readability.

The VOCIS model is able to proceed to non-linear regressions, based on Gnuplot fitting features [142]. In addition, multiple regressions are implemented for representing piecewise functions. In effect, many measures of interest present a saturation point, which is hardly represented by standard regressions. For instance, let us consider the exponential saturation formula for a data set $\{x_i, 1 \leq i \leq N\}$, in Equation 4.3. Since the switch index $I$ is unknown in general, the regression of the data set to this expression roughly requires to try the regression of this expression once for each value (i.e. $\forall I \in \{2, N-1\}$).

$$f(x_i) = \begin{cases} a + bx_i & \text{if } i \leq I \\ a + bx_I + b|c|\left(1 - e^{\frac{x_I - x_i}{|c|}}\right) \text{ (saturation)} & \text{if } i > I \end{cases} \tag{4.3}$$

In addition, multiple regression models may be compared for a given data set. In order to devise which is the most appropriate, the WSSR[16] of each regression is utilized. This value reflects the inaccuracy of a given regression compared to actual data, weighted by the standard deviation of each data point. As a consequence, regressions that output the smallest WSSR are selected. More advanced statistical criteria based on the regression residue would provide a more accurate fitting [143], yet the WSSR provides a decent choice.

[16] Weighted Sum of Square Residue

## 4.4 Measurements under interconnect defects

The measurement of fault tolerance and performance in a faulty network is a challenging task. In effect, fault patterns and lesser configuration attributes may completely redraw the resulting picture. In Section 4.4.1, a rigorous measurements timing is proposed in order to avoid start-up and termination artifacts. Then, a classification of packet loss' causes is proposed in Section 4.4.2. Finally, a discussion on the evolution of performance in the presence of interconnect defects is given in Section 4.4.3.

### 4.4.1 Measurement timing

The proceedings of fault tolerance simulations follow the phases displayed in Figure 4.7, in order to provide proper results. First of all, a warming period is required for the interconnect to reach its nominal traffic. In addition, this warming period may be utilized to inject permanent faults before the measurements start, and avoid sudden perturbation in the performance readings. After warming is over, the stabilization phase starts. Transient and intermittent faults are injected during this period. Stabilization ends either after a predefined time period or a premature ending event occurred. For instance, the saturation of queue buffers in the case of queue NIC exhaustion policy triggers the end of this stage. Upon the end of stabilization, the performance-related measurements and normal packet injection cease. Then, the model enters a draining phase, where packets remaining in the interconnect are given *enough time* to reach their respective destination. If necessary, the retransmission of packets and Acks is allowed, not to distort the fault tolerance measures. Finally, the interconnect simulation ends, and remaining in-transit packets are considered as lost.

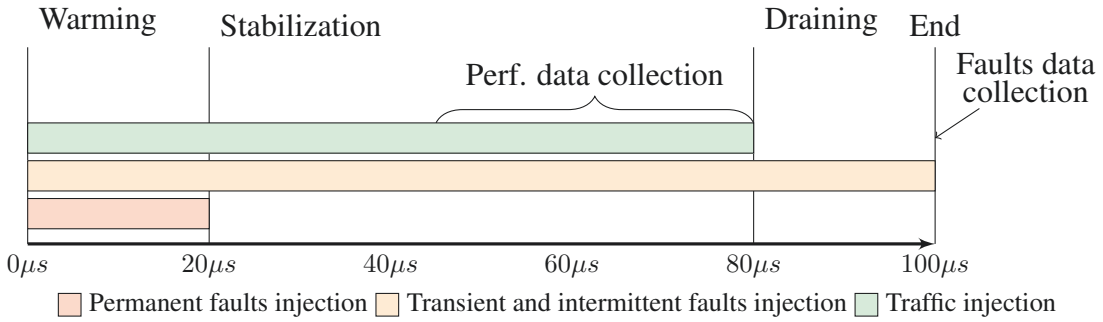



Figure 4.7: Simulation procedure for both performance and fault tolerance measurements

### 4.4.2 Packet loss root analysis

In a faulty NoC, the loss of a packet may occur for different reasons. Of course, a limitation of the routing algorithm may cause such a loss, but there are other causes related to the simulation itself, which deserve a brief explanation.

#### Intrinsic losses

Intrinsic losses are directly related to the fault and message injection settings. This class regroups source, destination and partitioning losses. A **source loss** occurs when the source node and/or router of a packet is faulty. This loss type does not impact the interconnect since no flit could possibly enter healthy interconnect resources. A **destination loss** happens if the destination node and/or router of a packet is faulty. These losses are frequent even for low fault rates, and often reduce drastically the

interconnect performance, following discussion in Section 4.2.4. Hence, researchers usually prefer packet injection algorithms that discard faulty destinations. **Partitioning losses** occur for high fault rates, when there exist no fault-free route between the source and target node. During first-stage simulations, partitioning may be avoided by the simulator to avoid difficult cases. However, simulating only non-partitioned interconnects may bias significantly the final results, especially for high defect rates.

#### Partial losses

Partial losses are impacted by the routing algorithm in a minor proportion. In effect, the probability of such losses is proportional to the duration of packets routing (i.e. latency). Hence, the performance of the routing algorithm impacts the rate of partial losses. **Network losses** occur when the network resource that last stored the packet's head flit is faulty. There are two interpretations to these losses. If the routing algorithm is fault-oblivious (or buggy), the routing algorithm could be charged for network losses, since it directed the packet into an already faulty resource. Otherwise, the routing algorithm should not be blamed for this loss, since the fault arguably occured after the head flit entered this resource. In the presence of transient faults, **corruption losses** happen when one or more flits of the packet were corrupted. These losses may occur either during routing with HBH transmission schemes or upon reception at the destination node for ETE schemes.

#### Routing losses

Finally, a **routing loss** occurs when a packet is lost while none of the previous loss conditions were met. These losses characterize the fault tolerance of fault-aware routing algorithms.

### 4.4.3 Performance in the presence of defects

The fluctuation of NoC performance metrics in the presence of faults often requires a thorough analysis. On one side, the throughput degradation is strongly related to the traffic pattern, and more specifically to the average source-destination distance. On the other side, the average latency trends are even more complex, and depend on the network utilization rate and routing complexity.

The degradation of performance in a network subject to defects essentially consists in 3 phases. A linear degradation phase, where performance decreases smoothly as the interconnect exploits slack resources and balances the traffic. Then, network saturation occurs when interconnect resources are too scarce. In this critical mode, a minute event (e.g. a long packet contention) generates significant degradation. Finally, a "desertification" phase occurs when most routes hampered by faults, and the routing algorithm is not able to transfer many packets. In effect, most NIC[17] are waiting for the timer attached to lost packets to expire.

The silence of lost packets is an important bias for the measurement of performance in the presence of faults. In effect, aborted message transfers do not enter into the throughput or latency values. As a matter of fact, a routing algorithm that drops most of long-distance packets may show better performance results than an ideal routing algorithm transferring as much packets as possible. While throughput partially reflects the loss of long-distance packets, the latency is indeed smaller for simpler routing algorithms since most transferred packets use a short route, weakly affected by faults.

Another issue that should be considered is the impact of timeouts in a faulty interconnect. In effect, when a packet is lost, the source NIC is usually blocked until a timeout triggers the retransmission

[17] Network Interface Circuit

packet retransmission. This ETE scheme is required to cope with permanent faults in the interconnect. However such timeout-based retransmission severely impacts the overall performance. Since the time-out value is generally one order of magnitude larger than average packets latency to avoid unnecessary reemissions, a single packet loss potentially delays the transmission of dizains of packets.

## 4.5 Conclusions and future works

In this chapter, the VOCIS[18] model was presented with a research perspective. First, the different mechanisms offering genericity to the model were described in Section 4.2. Then, the practical usage of this model was discussed in Section 4.3, and finally some experiences on the simulation of defective interconnects are given.

Overall, the proposed model offers a generic base to simulate heterogeneous on-chip interconnects, with a minimal redesign effort. This capacity is particularly important when different fault-tolerant routing algorithms need to be compared. In addition, the advanced visualization features included in VOCIS allow for a better understanding of the internal dynamics of defective interconnects. In our view, the combination of a highly generic core and advanced visualization features improves the efficiency of users by reducing development and debug delays.

[18] Versatile On-Chip Interconnect Simulation model

# Chapter 5

# Conclusions and future works

In a future of uncertainties, the advent of unreliable many-core processors is very likely. This paradigm will seemingly offer unprecedented performance with limited energy requirements. However, this paradigm is paved with a large number of technical challenges. In particular, offering sufficient resiliency and performance for parallel applications is a tedious task. As a consequence, their implementation will certainly be based on a combination of fault-tolerant techniques operating at multiple levels.

In this thesis, we proposed two fundamental elements for the implementation of unreliable many-core processors. A fault-tolerant routing algorithm for exploiting the core interconnect in the presence of defects, and a fault-tolerant self-adaptive framework for parallel applications to execute on these targets without the hassles of fault tolerance and variability support.

In addition, a generic model was developed and enabled novel visualization features. This simulator is offer a rare solution to compare heterogeneous interconnect solutions, while offering a fine grain analysis of their internal behaviour.

## 5.1 Highly fault-tolerant routing algorithms for large interconnects

In Chapter 2, a set of fault-tolerant routing algorithms were presented. These algorithms provide a tunable level of fault tolerance, depending on the application requirements. For instance, the Variant A algorithm, presented in Section 2.4.2, provides a limited amount of fault tolerance, but does not have large silicon requirements. At the opposite, the Variant C (shown in Section 2.4.4) is able to transmit packets as long as a fault-free route exists, but the overhead for VS[1] buffers is significant. The proposed routing algorithms are combined with a dynamic recovery mechanism, that ensures the recovery of the interconnect upon the occurrence of permanent and transient faults.

The performance of the interconnect is improved by the implementation of explicit routes. In effect, most applications transmit several messages from the same source node to the same destination. The explicit route mechanism reduces the length of packets' route in the presence of defects, and consequently improves the achievable throughput.

As a future work, another routing algorithm variant will be developed, based on the Variant C. This algorithm will provide a similar fault tolerance level, with the addition of multicast support and improved

[1] Virtual Source

performance.

## 5.2 Fault-tolerant self-adaptive applications for many-core chips

In Chapter 3, a fault-tolerant framework for self-adaptive applications is proposed. This approach allows for the opportunistic execution of applications, based on the dynamic mapping of tasks.

First, a compliant application model was proposed, and the fault tolerance of the proposed scheme was explained. This hierarchical model ensures that final application outputs are computed, despite the apparition of run-time defects.

Second, a fault-tolerant mapping mechanism was discussed. This dynamic approach is able to select suitable nodes for each task, at run-time. In order to maximize performance and resilience of the application, interacting tasks are mapped on close nodes.

Third, the fault-tolerant mapping was augmented with a variability-aware capacity. Hence, applications are able to adaptively select nodes that reduce the overall energy for each task, in the presence of high variability.

These proposition were supported by various simulations, which show the feasibility of this approach and the efficiency of a streaming application.

## 5.3 A Versatile On-Chip Interconnect model for large defective NoCs[2] simulation

In Chapter 4, the VOCIS[3] simulator is proposed for generic fault tolerance experiments. This model exhibit several unique functionalities that offer rare visualization opportunities to the researcher.

First, the architecture of the VOCIS model is described, and specificities of interest are detailed. For instance, the intrinsic support of multicast and adaptive routing is discussed in Section 4.2.3. Moreover, the details of the infrastructure that permit generic fault tolerance experiments are described.

Second, VOCIS supports advanced visualization features such as 3D viewing and the tuning of logging verbosity at run-time. In addition, a distributed simulation framework based on the model was presented, and provides system resilience and consistency.

Finally, the results of a few NoCs experiences were presented, to highlight the complexity of NoCs behavior in the presence of defects. For instance, the analysis of packets loss requires a thorough knowledge of the interconnect state and model behavior.

As a future work, state-of-the-art propositions will be implemented in this model, in order to provide quantitative comparison against our contributions. In addition, the efficiency of visualization will be improved to support thousand nodes interconnects.

[2] Network on Chips
[3] Versatile On-Chip Interconnect Simulation model

## 5.4 A global research perspective

In the future, unreliable many-core processors will probably reach a large market, since they will provide high flexibility and energy efficiency. A key aspect of this approach is that the full potential of the most aggressive technologies -prone to a large variability- will be exploited. Indeed, providing fault-tolerant mechanism mechanism alleviates the need for guardbanding and strict testing of chips.

However, this appealing scenario requires the construction of an holistic framework that effectively protects the application from the sporadic apparition of faults in the processor. In [10], the *Cells* project proposes such a framework, based on fault-tolerant mechanisms at multiple layers (e.g. devices, circuits, software). Hence, the enforcement of unreliable many-core processors will require a huge effort for coordinating disparate techniques efficiently, and ultimately run applications obliviously of the many faults appearing in the executing processor. In this respect, the exploitation of unreliable many-core processors will require fault-tolerance techniques both at the interconnect level and the computation node level. This thesis discussed fundamental issues for both levels.

At the interconnect level, different routing algorithms have been proposed and tested with a high-level model. Each algorithm proposes a different trade-off between the router complexity and the rate of packets transmission under defects. In fact, the actual fault tolerance of interconnects will not only be affected by the ability of routing algorithms to bypass faulty resources, but also by the probability that individual routers and links fail. As a consequence, complex routing algorithm may improve the interconnect resiliency only when its overhead does not affect the router failure rate. Finally, depending on the target technology, simple routing algorithms may be preferred for reducing the silicon and latency overheads (and therefore the router failure rate); or complex routing algorithms may be called to bypass multiple defective links and routers. This dilemma gives an added value to the work produced in this thesis, since Variant A, B and C have gradual requirements, to cope with a wide variety of situations.

At the computation node level, a software framework will be required for relieving developers from the burden of managing the effects of faults. These frameworks will necessarily react to new faults with tasks re-allocation and/or message reemission, since neither interconnect nor nodes are able to guarantee the success of messages transfer and task execution, respectively. Additionally, frameworks will also conduct proactive actions in order to minimize the apparition of new faults and/or their impact on the application. For instance, a framework may avoid the apparition of hotspots, effectively reducing the apparition of temperature-dependent faults. Alternatively, mapping communicating tasks close form each other reduces the number of hops required for messages transfer, and consequently the number of -unreliable- routers.

Last but not least, the modeling and analysis of these extremely complex chips will be hindered by the multiplicity of conceivable fault patterns. Glancing at the challenges currently prompted by the shift to parallel applications gives a insight of the chaos when applications will be executed on massively unreliable processors. Conceivably, designers and developers will strive to understand how specific fault scenarios might cause the crash of an application. Again based on the trends on parallelization, substantial improvements will be required in the debugging and visualization of the execution of applications. In this respect, visual models -such as VOCIS- may offer a dense yet readable view of the internal state of processors. In effect, the huge amount of logging data requires preprocessing mechanisms for providing developers (and designers) with an understandable representation of the situation at hands. Hence, the advent of unreliable many-core processors will likely come with a new generation of modeling and debugging tools, with more and more visual content.

# List of publications

- G. Bizot, **F. Chaix**, N-E. Zergainoh and M. Nicolaidis, *Variability-Aware and Fault-tolerant Self-Adaptive applications for Many-Core chips*, IOLTS'13
- G. Bizot, D. Avresky, **F. Chaix**, N-E. Zergainoh and M. Nicolaidis, *Self-Recovering Parallel Applications in Multi-Core Systems*, NCA'11
- **F. Chaix**, G. Bizot, M. Nicolaidis and N-E. Zergainoh, *Variability-aware Task mapping strategies for Many-cores processor chips*, IOLTS'11
- **F. Chaix**, D. Avresky, N-E. Zergainoh and M. Nicolaidis, *A fault-tolerant deadlock-free adaptive routing for On Chip interconnects*, DATE'11
- **F. Chaix**, D. Avresky, N-E. Zergainoh and M. Nicolaidis, *Fault-tolerant deadlock-free adaptive routing for any set of link and node failures in Multi-Cores systems*, NCA'10

# Appendix A

# Annexes

## A.1 Variability modeling

In this thesis, the modeling of variability is based on the VARIUS model presented in [106]. However, the uncertainties on the evolution of the variability in future technology discouraged the utilization of a single chip design following this model. As a consequence, a synthetic floorplan was assumed, and different variability scenarios were utilised following Table 3.1.

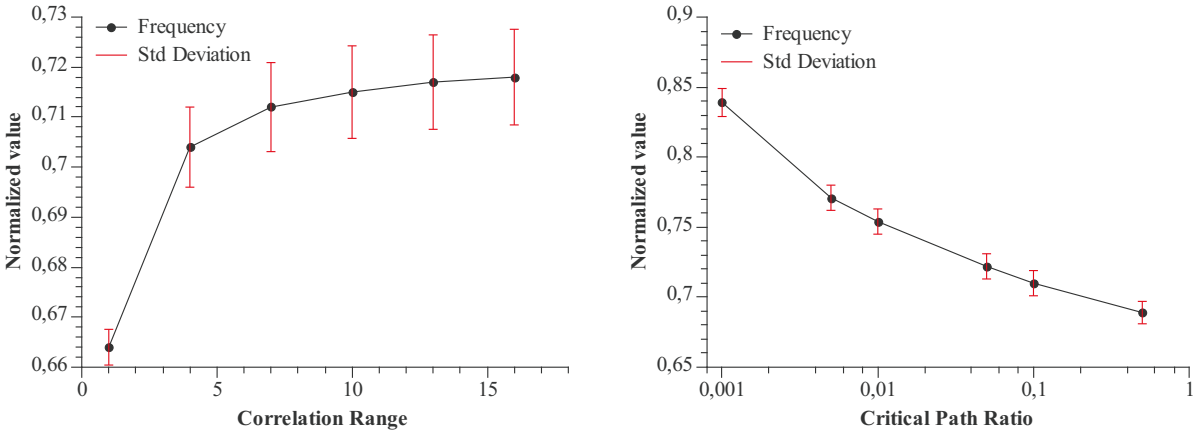
The process variations mostly affect the transistors threshold voltage $V_{th}$ of transistors and usually consist of the systematic and the random variations. Both random and systematic variations follow an independent normal distribution of 0-mean and standard deviation of respectively $\sigma_{rand}$ and $\sigma_{syst}$. In addition, the value of the systematic variation of two different transistors is correlated by a factor $\rho(r)$, depending on the distance $r$ between them, according to Equation A.1. The $\phi$ parameter is the correlation range.

$$\rho(r) = \begin{cases} 1 - \frac{3r}{2\phi} + \frac{r^3}{2\phi^3} & r \leqslant \phi \\ 0 & else \end{cases} \tag{A.1}$$

The expression of the transistors' threshold voltage $V_{th}$ is given in Equation A.2, where $\overline{V_{th}}$ is the average value of the threshold voltage.

$$V_{th} = \overline{V_{th}} + \mathcal{N}(0, \sigma_{rand}) + \mathcal{N}(0, \sigma_{syst}, \rho) \tag{A.2}$$

The impact of variability depends on many parameters directly related to the design and manufacturing process. For instance, in Figure A.1, the impact of the ratio of critical paths amongst core paths $r_{cp}$ and the correlation range $\phi$ for the systematic variations are shown, under Case 5 variability scenario, with $r_{cp0} = 0.1$, $\phi_0 = 6[cores]$ and $V_{dd} = 1.2[V]$.




(a) Frequency vs. correlation range

(b) Frequency vs. critical path ratio

Figure A.1: Impact of model parameters on the cores frequency

# Glossary

**deadlock freedom** Absence of deadlocks, in particular regarding wormhole routing algorithm, c.f. Section 2.1.3 . 107

**fault tolerance** The ability for a system to maintain its functionality despite the occurence of faults. 107

**flit** A portion of packet used in wormhole switching networks. c.f. Section 2.1.3. ix, xii, xxvii, 11–13, 15, 16, 19, 26, 28, 29, 31, 46, 48–50, 63, 73–75, 78, 107

**many-core** A processor consisting of a large number of processing cores.. xxv, xxviii, 5, 7–9, 25, 53–55, 57–59, 65, 72, 75, 80, 99–101, 107

**message passing** A processor architecture where cores exchange data through the exchange of messages.. 6, 13, 20, 55, 107

**multi-core** A processor consisting of a few processing cores.. 1, 5, 6, 28, 51, 55, 58, 66, 107

**neighbor node** In a 2D-Mesh topology, a node distant from node $n$ by exactly one hop. 107

**shared memory** A processor architecture where computation cores access to a common memory space.. 6, 13, 55, 107

**XY** In 2D-Mesh interconnects, XY is a classic deadlock-free routing algorithm that first advances on the horizontal axis (X) until it reaches the column of target node, and then progress on the vertical direction (Y).. 16, 17, 23, 107

# Acronyms

**Ack** A positive acknowledgement is used to validate the reception of corresponding data. 20, 34, 44, 49, 84, 107

**BIST** The inclusion of circuits dedicated for the test of the system and its featuring components. x, 19, 28, 60, 107

**CISC** A family of computation cores including complex operations that require multiple clock cycles and often microcode.. 107

**CMOS** The mainstream microelectronic etching process, in which gates are built by associating complementary transistors. 107

**DAG** A directed graph (i.e. with edges) and no cycle.. xi, xvi, xxviii, 56, 63, 66, 74, 80, 107

**DVFS** Adaptation of the operating frequency and voltage of a chip to current conditions. x, 5, 59, 60, 62, 107

**ECC** The redundant encoding of information of interest, in order to counter the later corruption of data.. 19, 28, 29, 107

**EDA** The industry providing programs for aiding the creation and realization of integrated and discrete electronic circuits. 6, 107

**ETE** Describes a technique acting solely upon start and ending of a process, e.g. the emission and reception of a message.. 13, 15, 19–21, 107

**FEC** The correction of some protected data without delaying the transmission of these data. 19, 107

**FIFO** Data structure where the first value pushed in is the first to be popped out.. xi, 107

**GALS** A paradigm where chips are composed of a set of synchronous circuits (or islands), which each utilize a specific clock signal.. x, 59, 60, 107

**GUI** In a software application, the set of components that permit the interaction between an human user and the application core. 83, 89, 107

**HBH** Describes a technique acting at each packet hop, allowing faster effects.. 19, 20, 107

**HPC** Computers built specifically for the realization of computation-intensive tasks. 10, 107

**IAM** A message used for asserting the liveness of a remote computation core.. xi, xii, xvii, 64, 66, 67, 107

**IAP** A message used for asserting the liveness of an interconnection resource.. 19, 107

**IC** An electronic circuit consisting of microscopic devices usually fabricated over a silicon wafer. 3, 10, 107

**IDWT** A transformation that converts a wavelet representation of data into a spatial representation.. 74, 107

**IP** A circuit designed by a company.. viii, 5, 10, 11, 107

**MPPA** A type of processor consisting of hundreds or even thousands of primitive cores, tied by a stream-oriented interconnect. 5, 107

**MTBF** The statistical period between two failures of a system.. 57, 107

**MTS** The maximum size of a packet in a given network.. 13, 107

**Nack** A positive acknowledgement is used to notify the loss of corresponding data. xii, 20, 28, 29, 34, 49, 66, 107

**NBTI** A physical effect that degrades the function of transistors, and worsens as temperature increases. A partial recovery of this effect may occur over time, once stress conditions ended. 4, 18, 107

**NIC** A circuit dedicated to the conversion of computation cores requests into interconnection packets and *vice-versa*.. viii, xx, 7, 11, 13–15, 19, 20, 22, 25, 26, 29, 34, 35, 46, 47, 49, 54, 60, 84, 87, 96, 107

**NoC** An interconnect structure mimicking computer networks. viii, ix, xi, xiii, xviii–xxi, xxv, xxvii, 6–16, 18–22, 24, 25, 30, 51, 56, 58–60, 67, 82, 88, 100, 107

**OS** A software dedicated to the exploitation of hardware resources by third-party applications. 6, 55, 58, 66, 70, 82, 107

**QoS** An user-oriented measurement of the performance of a system, and by extension, the support of this performance in the presence of adverse events. 12, 107

**RDF** The physical effect observed in latest etching processes, which inject too unevenly dopant atoms, thus causing variations in the resulting characteristics. 4, 107

**RISC** A family of computation cores limited to simple operations executed in one clock cycle. 107

**RNack** A positive acknowledgement is used to notify the loss of corresponding data, due to a temporary defect.. 20, 107

**RS** A specific mechanism used to break potential deadlocks in the proposed routing algorithms.. xiii, xiv, 36, 38, 39, 44, 50, 52, 107

**RTL** The abstraction level at which an integrated circuit is described by its registers and the connexion between them.. 81, 82, 107

**SoC** A complex association of different circuits on the same chip. 10, 14, 54, 107

**TSV** Metal bridge from one silicon die to another die, which traverses completely one die.. 14, 20, 107

**TTL** A numeric quantity utilized to limit the propagation of a packet. 18, 107

**UID** A numeric value that uniquely defines an object.. 84, 89, 107

**UNack** A positive acknowledgement is used to notify the loss of corresponding data, due to an unreachable target node.. 20, 107

**VC** A sub-division of routers' physical channels. ix, xi, 10, 12, 15–17, 24, 25, 29, 30, 32, 46–48, 84–86, 107

**VCT** An evolution of packet-switching networks, where packets are sliced into flits.. 15, 24, 107

**VN** A virtual repartition of VCs into networks. xi–xiii, xxvii, 22, 23, 30–34, 36, 37, 46, 47, 50, 52, 107

**VOCIS** The interconnection simulation model proposed in this thesis.. x, xi, xix–xxi, 8, 81, 85, 86, 89, 91, 97, 100, 101, 107

**VS** A specific mechanism used to break potential deadlocks in the proposed routing algorithms.. xi, xiii, xxvii, 25–27, 29, 36–39, 41, 42, 45, 48, 50–52, 84, 99, 107

**WSSR** A measure of the accuracy of a data regression.. 94, 107

# Bibliography


[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," *Micro, IEEE*, vol. 32, no. 3, pp. 122–134, may-june 2012. iv, 4

[2] L. Benini and G. D. Micheli, "Networks on Chips: a new SoC paradigm," *Computer*, 2002. iv, 10

[3] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks* . Elsiever, 2004. v, 15

[4] E. Beigne, F. Clermidy, S. Miermont, and P. Vivet, "Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, Apr. 2008, pp. 129–138. vi, 60

[5] F. Chaix, D. Avresky, N. Zergainoh, and M. Nicolaidis, " Fault-tolerant deadlock-free adaptive routing for any set of link and node failures in Multi-Cores systems," in *IEEE International Symposium on Network Computing and Applications*, july 2010. viii, 9

[6] ——, " A fault-tolerant deadlock-free adaptive routing for On Chip interconnects," in *Design, Automation and Test in Europe Conference and Exhibition*, march 2011. viii, x, 9

[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture: a system of patterns*. New York, NY, USA: John Wiley & Sons, Inc., 1996. xii, 56

[8] F. Chaix, G. Bizot, M. Nicolaidis, and N. Zergainoh, " Variability-aware Task mapping strategies for Many-cores processor chips," in *International On-Line Testing Symposium*, july 2011. xiii, 53

[9] The Green500. [Online]. Available: http://www.green500.org 3

[10] M. Nicolaidis, "Biologically Inspired Robust Tera-Device Processors," *Design Test of Computers, IEEE*, vol. 29, no. 5, pp. 94–99, Oct. 7, 101

[11] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, Jan. 2009. 9

[12] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *DAC'01: Proc. of the 38th Conf. on Design Automation*, 2001. 10

[13] E. Salminen, A. Kulmala, and T. D. Hamalainen, "Survey of Network-on-chip Proposals," *OCP-IP white paper*, 2008. 10

[14] S. Bell et al, "TILE64 Processor: A 64-Core SoC with Mesh Interconnect," *ISSCC*, 2008. 11

[15] Kalray's Multi-Purpose Processor Array 256. [Online]. Available: http://www.kalray.eu/products/mppa-manycore/mppa-256/ 11

[16] Adapteva's Epiphany IP. [Online]. Available: http://www.adapteva.com/products/epiphany-ip/epiphany-architecture-ip/ 11

[17] C. Wang, W.-H. Hu, and N. Bagherzadeh, "A Wireless Network-on-Chip Design for Multicore Platforms," in *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, Feb. 2011, pp. 409–416. 11

[18] H. Matsutani, P. Bogdan, R. Marculescu, Y. Take, D. Sasaki, H. Zhang, M. Koibuchi, T. Kuroda, and H. Amano, *ASP-DAC*. 11

[19] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Neighbors-on-Path: A New Selection Strategy for On-Chip Networks," in *Embedded Systems for Real Time Multimedia, Proceedings of the 2006 IEEE/ACM/IFIP Workshop on*, 2006, pp. 79–84. 12

[20] D. Matos, C. Concatto, M. Kreutz, F. Kastensmidt, L. Carro, and A. Susin, "Reconfigurable Routers for Low Power and High Performance," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 11, pp. 2045–2057, nov. 2011. 12

[21] P. Ghosh, A. Ravi, and A. Sen, "An Analytical Framework with Bounded Deflection Adaptive Routing for Networks-on-Chip," jul. 2010. 12, 24

[22] M. Radetzki and A. Kohler, "An intelligent deflection router for networks-on-chip," *Intelligent solutions in Embedded Systems, 2009 Seventh Workshop on*, jun. 2009. 12, 24

[23] J. Lee, D. Lee, S. Kim, and K. Choi, "Deflection Routing in 3D Network-on-Chip with TSV Serialization," *ASP-DAC*, 2013. 12, 24

[24] F. Jafari, M. Talebi, A. Khonsari, and M. Yaghmaee, "A Novel Congestion Control Scheme in Network-on-Chip Based on Best Effort Delay-Sum Optimization," May 2008. 14

[25] J. van den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," April 2007. 14

[26] W. Zhong, B. Yu, S. Chen, T. Yoshimura, S. Dong, and S. Goto, "Application-specific Network-on-Chip synthesis: Cluster generation and network component insertion," in *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, march 2011, pp. 1–6. 14

[27] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application Specific Routing Algorithms for Networks on Chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, March 2009. 14, 21

[28] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad, "Application-Aware Topology Reconfiguration for On-Chip Networks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 11, pp. 2010–2022, nov. 2011. 14

[29] C. Wang, W.-H. Hu, and N. Bagherzadeh, "Congestion-aware Network-on-Chip router architecture," in *Computer Architecture and Digital Systems (CADS), 2010 15th CSI International Symposium on*, Sep. 2010. 14

[30] M. Moadeli, P. Maji, and W. Vanderbauwhede, "Design and implementation of the Quarc Network on-Chip," may. 2009. 15

[31] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. 15

[32] A. Roca, J. Flieh, F. Silla, and J. Duato, "VCTlite: Towards an efficient implementation of virtual cut-through switching in on-chip networks," in *High Performance Computing (HiPC), 2010 International Conference on*, Dec. 2010. 15

[33] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, Sept.-Oct. 2005. 16

[34] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, 2010, pp. 306–311. 16

[35] W. Dally and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *Computers, IEEE Transactions on*, vol. C-36, no. 5, pp. 547–553, 1987. 17

[36] J. Duato, "A theory of fault-tolerant routing in wormhole networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 8, Aug 1997. 17

[37] C. J. Glass and L. M. Ni., "The turn model for adaptive routing," *ACM SIGARCH Computer Architecture News*, 1992. 17, 21

[38] F. Verbeek and J. Schmaltz, "Formal specification of networks-on-chips: deadlock and evacuation," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, Mar. 2010. 17

[39] ——, "A Fast and Verified Algorithm for Proving Store-and-Forward Networks Deadlock-Free," in *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, Feb. 2011, pp. 3–10. 17

[40] C. Seiculescu, S. Murali, L. Benini, and G. De Micheli, "A method to remove deadlocks in Networks-on-Chips with Wormhole flow control," mar. 2010. 17

[41] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. Das, "Exploring Fault-Tolerant Network-on-Chip Architectures," *International Conference on Dependable Systems and Networks*, June 2006. 17, 20

[42] C. Hernandez, F. Silla, and J. Duato, "A methodology for the characterization of process variation in NoC links," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, Mar. 2010. 18

[43] K. Aisopos, C.-H. O. Chen, and L.-S. Peh, "Enabling system-level modeling of variation-induced faults in Networks-on-Chips," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, june 2011, pp. 930–935. 18

[44] C. Nicopoulos, S. Srinivasan, A. Yanamandra, D. Park, V. Narayanan, C. R. Das, and M. J. Irwin, "On the Effects of Process Variation in Network-on-Chip Architectures," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 3, jul. 2010. 18

[45] M. Nicolaidis, "GRAAL: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies," in *IEEE International Test Conference*, Oct. 2007. 19, 57, 60

[46] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester, "Vicis: A reliable network for unreliable silicon," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, 2009, pp. 812–817. 19

[47] M. Koibuchi, H. Matsutani, H. Amano, and T. Mark Pinkston, "A Lightweight Fault-Tolerant Mechanism for Network-on-Chip," in *ACM/IEEE International Symposium on Networks-on-Chip*, April 2008. 19, 23

[48] S. E. Lee, Y. S. Yang, G. Choi, W. Wu, and R. Iyer, "Low-Power, Resilient Interconnection with Orthogonal Latin Squares," *Design Test of Computers, IEEE*, vol. 28, no. 2, pp. 30–39, march-april 2011. 19

[49] W.-C. Tsai, D.-Y. Zheng, S.-J. Chen, and Y.-H. Hu, "A Fault-Tolerant NoC Scheme using bidirectional channel," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, june 2011, pp. 918–923. 19

[50] M. Nicolaidis, V. Pasca, and L. Anghel, "I-BIRAS: Interconnect Built-In Self-Repair and Adaptive Serialization in 3D Integrated Systems," in *European Test Symposium (ETS), 2011 16th IEEE*, 2011, pp. 208–208. 20

[51] A. Kologeski, C. Concatto, L. Carro, and F. L. Kastensmidt, "Adaptive approach to tolerate multiple faulty links in Network-on-Chip," in *Test Workshop (LATW), 2011 12th Latin American*, march 2011, pp. 1–6. 20

[52] C. Grecu, L. Anghel, P. Pande, A. Ivanov, and R. Saleh, "Essential Fault-Tolerance Metrics for NoC Infrastructures," July 2007. 20

[53] H. Kariniemi and J. Nurmi, "NoC Interface for fault-tolerant Message-Passing communication on Multiprocessor SoC platform," in *NORCHIP, 2009*, Nov. 2009. 20

[54] T. Mak, K. Lam, P. Cheung, and W. Luk, "Adaptive Routing in Network-on-Chips Using a Dynamic Programming Network," *Industrial Electronics, IEEE Transactions on*, vol. PP, no. 99, 2010. 21

[55] Y. Jin, E. J. Kim, and T. M. Pinkston, "Communication-Aware Globally-Coordinated On-Chip Networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 242–254, feb. 2012. 21

[56] J. Flich, T. Skeie, A. Mejia, O. Lysne, P. Lopez, A. Robles, J. Duato, M. Koibuchi, T. Rokicki, and J. Sancho, "A Survey and Evaluation of Topology-Agnostic Deterministic Routing Algorithms," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 3, pp. 405–425, march 2012. 21

[57] G.-M. Chiu, "The odd-even turn model for adaptive routing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 11, no. 7, Jul 2000. 21

[58] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie, "Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, April 2006. 22

[59] J. Flich, S. Rodrigo *et al.*, "An Efficient Implementation of Distributed Routing Algorithms for NoCs," *ACM/IEEE International Symposium on Networks-on-Chip*, April 2008. 22, 23

[60] P.-T. Huang and W. Hwang, "An adaptive congestion-aware routing algorithm for mesh network-on-chip platform," in *SOC Conference, 2009. SOCC 2009. IEEE International*, 9-11 2009. 22

[61] D. Wu, B. Al-Hashimi, and M. Schmitz, "Improving routing efficiency for network-on-chip through contention-aware input selection," jan. 2006. 22

[62] G. Ascia, V. Catania, M. Palesi, and D. Patti, "Implementation and Analysis of a New Selection Strategy for Adaptive Routing in Networks-on-Chip," *Computers, IEEE Transactions on*, vol. 57, no. 6, June 2008. 22

[63] P. Gratz, B. Grot, and S. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, Feb. 2008. 22

[64] J. Hu and R. Marculescu, "DyAD - smart routing for networks-on-chip," 2004. 22

[65] E.-J. Chang, C.-H. Chao, K.-Y. Jheng, H.-K. Hsin, and A.-Y. Wu, "ACO-based Cascaded Adaptive Routing for traffic balancing in NoC systems," in *Green Circuits and Systems (ICGCS), 2010 International Conference on*, Jun. 2010. 22

[66] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A Highly Resilient Routing Algorithm for Fault-Tolerant NoCs," in *Design, Automation and Test in Europe*, no. 2.2, April 2009. 22

[67] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, June 2008. 22

[68] S. Pasricha, Y. Zou, D. Connors, and H. J. Siegel, "OE+IOE: A novel turn model based fault tolerant routing scheme for networks-on-chip," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, oct. 2010, pp. 85–93. 22

[69] C. Cunningham and D. Avresky, "Fault-tolerant adaptive routing for two-dimensional meshes," in *IEEE Symposium on High-Performance Computer Architecture*, 1995. 22, 31

[70] M. Ebrahimi, M. Daneshtalab, J. Plosila, and F. Mehdipou, "MD: Minimal Path-based Approach for Fault-Tolerant Routing in On-Chip Networks," *ASP-DAC*, 2013. 23

[71] D. Xiang, "Deadlock-Free Adaptive Routing in Meshes with Fault-Tolerance Ability Based on Channel Overlapping," *Dependable and Secure Computing, IEEE Transactions on*, vol. 8, no. 1, Jan. 2011. 23

[72] S. Jovanovic, C. Tanougast, S. Weber, and C. Bobda, "A new deadlock-free fault-tolerant routing algorithm for NoC interconnections," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Sep. 2009. 23

[73] R. Boppana and S. Chalasani, "Fault-tolerant wormhole routing algorithms for mesh networks," *IEEE Transactions on Computers*, vol. 44, no. 7, Jul 1995. 23

[74] Y. Fukushima, M. Fukushi, and S. Horiguchi, "Fault-Tolerant Routing Algorithm for Network on Chip without Virtual Channels," in *Defect and Fault Tolerance in VLSI Systems, 2009. DFT '09. 24th IEEE International Symposium on*, 7-9 2009. 23

[75] A. Mortazavi and F. Safaei, "Dynamic fault-tolerant wormhole routing in 2-D meshes," in *Computer Architecture and Digital Systems (CADS), 2010 15th CSI International Symposium on*, Sep. 2010. 23

[76] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," *IEEE Computer society Annual Symposium on VLSI*, Feb. 2004. 24

[77] T. Dumitras, S. Kerner, and R. Marculescu, "Towards On-Chip Fault-Tolerant Communication," in *Proc. Asia & South Pacific Design Automation Conf. (ASP-DAC)*, January 2003. 24

[78] W. Song, D. Edwards, J. Nunez-Yanez, and S. Dasgupta, "Adaptive stochastic routing in fault-tolerant on-chip networks," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, May 2009. 24

[79] P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic Communication: A New Paradigm for Fault-Tolerant Networks-on-Chip," *VLSI Design*, 2007. 24

[80] H. Zhu, P. Pande, and C. Grecu, "Performance Evaluation of Adaptive Routing Algorithms for achieving Fault Tolerance in NoC Fabrics," in *Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on*, Jul. 2007. 24

[81] A. Kohler, G. Schley, and M. Radetzki, "Fault Tolerant Network on Chip Switching With Graceful Performance Degradation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, jun. 2010. 24

[82] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing," may. 2010. 24

[83] V. Puente, J. Gregorio, F. Vallejo, and R. Beivide, "Immunet: Dependable Routing for Interconnection Networks with Arbitrary Topology," *IEEE Transactions on Computers*, vol. 57, no. 12, Dec. 2008. 24

[84] D. Avresky and N. Natchev, "Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures," *IEEE Transactions on Computers*, vol. 54, no. 5, May 2005. 24

[85] C. Rusu, L. Anghel, and D. Avresky, "RILM: Reconfigurable inter-layer routing mechanism for 3D multi-layer networks-on-chip," in *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*, july 2010, pp. 121–126. 24

[86] A.-M. Rahmani, K. Latif, P. Liljeberg, J. Plosila, and H. Tenhunen, "A Stacked Mesh 3D NoC Architecture Enabling Congestion-Aware and Reliable Inter-layer Communication," in *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, Feb. 2011, pp. 423–430. 24

[87] S. Pasricha and Y. Zou, "A low overhead fault tolerant routing scheme for 3D Networks-on-Chip," in *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, march 2011, pp. 1–8. 24

[88] D.R. Avresky, C.M. Cunningham, and H. Ravichandran, "Fault-tolerant adaptive routing for two-dimensional meshes," *Int. Journal of Computer Systems Science and Engineering*, vol. 14, no. 6, november 1999. 37

[89] D. R. Avresky, Ed., *Dependable Network Computing*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. 41

[90] G. Bizot, D. Avresky, F. Chaix, N. Zergainoh, and M. Nicolaidis, " Self-Recovering Parallel Applications in Multi-Core Systems," in *IEEE International Symposium on Network Computing and Applications*, august 2011. 53

[91] G. Bizot, F. Chaix, N. Zergainoh, and M. Nicolaidis, "Variability-Aware and Fault-tolerant Self-Adaptive applications for Many-Core chips," in *IOLTS*, jul 2013. 53

[92] V. Nollet and D. Verkestt, "A Quick Safari Through the MPSoC Run-Time Management Jungle," *Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on*, Oct. 2007. 54

[93] Y. Lin, M. Kudlur, S. Mahlke, and T. Mudge, "Hierarchical coarse-grained stream compilation for software defined radio," in *CASES '07: Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*. New York, NY, USA: ACM, 2007, pp. 115–124. 56

[94] G. Shao, F. Berman, and R. Wolski, "Master/slave computing on the grid," in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, 2000, pp. 3–16. 56

[95] N. Ranaldo and E. Zimeo, "An economy-driven mapping heuristic for hierarchical master-slave applications in grid systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, apr 2006, p. 13 pp. 56

[96] H. C. Zhao, C. H. Xia, Z. Liu, and D. Towsley, "A unified modeling framework for distributed resource allocation of general fork and join processing networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 299–310, 2010. 56

[97] Q. Li, Y. Ruan, ShidaYang, and T. Jiang, "An optimal scheduling algorithm for fork-join task graphs," in *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, 2003, pp. 587–589. 56

[98] C.-L. Chou and R. Marculescu, "Incremental run-time application mapping for homogeneous nocs with multiple voltage levels," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, 30 2007-oct. 3 2007, pp. 161–166. 56

[99] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs," in *Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on*, may 2007, pp. 34–40. 56

[100] M. Mandelli, L. Ost, E. Carara, G. Guindani, T. Gouvea, G. Medeiros, and F. G. Moraes, "Energy-aware dynamic task mapping for noc-based mpsocs," in *International Symposium on Circuits and Systems*. IEEE, may 2011, pp. 1676–1679. 56

[101] E. de Souza Carvalho, N. Calazans, and F. Moraes, "Dynamic task mapping for mpsocs," *Design Test of Computers, IEEE*, vol. 27, no. 5, pp. 26–35, sep 2010. 56

[102] A. Mehran, A. Khademzadeh, and S. Saeidi, "Dsm: A heuristic dynamic spiral mapping algorithm for network on chip," *IEICE Electronics Express*, vol. 5, no. 13, pp. 464–471, 2008. 56

[103] P. Zipf, G. Sassatelli, N. Utlu, N. Saint-Jean, P. Benoit, and M. Glesner, "A decentralised task mapping approach for homogeneous multiprocessor network-on-chips," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 1–14, 2009. 56

[104] M. Al Faruque, R. Krist, and J. Henkel, "Adam: Run-time agent-based distributed application mapping for on-chip communication," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, jun 2008, pp. 760–765. 57

[105] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for MPSoC," *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, Nov. 2007. 57

[106] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 21, no. 1, Feb. 2008. 57, 61, 105

[107] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 2, Feb 2002. 57

[108] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar, "Within-Die Variation-Aware Dynamic-

Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, Jan. 2011. 57

[109] S. Majzoub, R. Saleh, S. Wilton, and R. Ward, "Energy Optimization for Many-Core Platforms: Communication and PVT Aware Voltage-Island Formation and Voltage Selection Algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 5, pp. 816–829, May 2010. 57

[110] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, 2008, pp. 363–374. 57

[111] S. Saha, "Modeling Process Variability in Scaled CMOS Technology," *Design Test of Computers, IEEE*, vol. 27, no. 2, pp. 8–16, 2010. 57

[112] D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, Feb. 2008. 57

[113] Z. Gu, C. Zhu, L. Shang, and R. Dick, "Application-Specific MPSoC Reliability Optimization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 5, May 2008. 57

[114] P. Zajac, " Fault Tolerance through Self-configuration in the future Nanoscale Multiprocessors," Ph.D. dissertation, JUNE 2008. 57

[115] J. H. Collet, P. Zajac, M. Psarakis, and D. Gizopoulos, "Chip Self-Organization and Fault Tolerance in Massively Defective Multicore Arrays," *Dependable and Secure Computing, IEEE Transactions on*, vol. 8, no. 2, Mar. 2011. 57

[116] R. Tornero, V. Sterrantino, M. Palesi, and J. M. Orduna, "A multi-objective strategy for concurrent mapping and routing in networks on chip," in *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8. 58

[117] M. Krstic, E. Grass, F. Gurkaynak, and P. Vivet, "Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook," *Design & Test of Computers, IEEE*, vol. 24, no. 5, Sept.-Oct. 2007. 59

[118] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart, "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, Jan. 2011. 59

[119] S. Miermont, " Controle distribue de la tension d'alimentation dans les architectures GALS et proposition d'un selecteur dynamique d'alimentation," Ph.D. dissertation. 60

[120] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie, "From Epidemics to Distributed Computing," *IEEE Computer*, vol. 37, no. 5, pp. 60–67, 2004. 67

[121] M. Jelasity and O. Babaoglu, "T-man: Gossip-based overlay topology management," in *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, Jul. 2005. 67

[122] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003. 67

[123] K. Jenkins, K. Hopkinson, and K. Birman, "A gossip protocol for subgroup multicast," *Distributed Computing Systems Workshops, International Conference on*, vol. 0, p. 0025, 2001. 67

[124] M. Drosg, "Frequency and probability distributions," in *Dealing with Uncertainties*. Springer Berlin Heidelberg, 2007, pp. 51–69, 10.1007/978-3-540-29608-9_5. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-29608-9_5 71

[125] "Jpeg 2000 image coding system (jpeg 2000 part i final committee draft version 1.0)," mar 2000. [Online]. Available: http://www.jpeg.org/jpeg2000/CDs15444.html 73, 74

[126] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718 82

[127] F. Fazzino, M. Palesi, and D. Patti. (2008) Noxim: Network-on-Chip Simulator. [Online]. Available: http://noxim.sourceforge.net 82, 84

[128] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny, "HNOCS: Modular open-source simulator for Heterogeneous NoCs," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, 2012, pp. 51–57. 82

[129] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, Apr. 2009. 82

[130] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005. [Online]. Available: http://doi.acm.org/10.1145/1105734.1105747 82

[131] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002. 82

[132] M. Hosseinabady and J. Nunez-Yanez, "Effective modelling of large NoCs using SystemC," may. 2010. 82

[133] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A Power-Area Simulator for Interconnection Networks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 1, pp. 191–196, jan. 2012. 82

[134] K. Skadron et al, "HotSpot: Techniques for Modeling Thermal Effects at the Processor-Architecture Level," *THERMINIC*, 2002. 82

[135] The Boost library . [Online]. Available: http://www.boost.org/ 83

[136] VTK: The Visualization ToolKit. [Online]. Available: http://www.vtk.org/ 83

[137] P. Abad, V. Puente, J. A. Gregorio, and P. Prieto, "Rotary router: an efficient architecture for CMP interconnection networks," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 116–125. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250678 86

[138] T. Pionteck, C. Osterloh, and C. Albrecht, "Latency reduction of selected data streams in Network-on-Chips for adaptive manycore systems," in *NORCHIP, 2010*, Nov. 2010. 86

[139] V. Shurbanov, D. Avresky, and R. Horst, "The effect of the router arbitration policy on the scalability of ServerNetTM topologies," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, Mar. 1998. 86

[140] B. Horst, D. Avresky, R. Wilkinson, D. Jewett, W. Watson, L. Young, and C. Cunningham, "Performance modeling of ServerNet topologies," in *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, Apr. 1996. 87

[141] D. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley. 92

[142] Gnuplot. [Online]. Available: http://www.gnuplot.info/ 94

[143] P. Cornillon and E. Matzner-Lober, *Régression, Théorie et application*. Springer. 94

***Abstract***—The perspective of nanometric technologies foreshadows the advent of processors consisting of hundreds of computation cores. However, the exploitation of these processors will require to cope with reliability and variability issues inherent to these aggressive manufacturing processes. In this thesis, we present a coherent set of techniques for the utilization of many-cores processors subject to high defect and variability rates.

First, the interconnection network reliability is addressed, with the presentation of several deadlock-free fault-tolerant routing algorithms, without routing tables for improving their scalability. The different variants of these algorithms allow for the tune-up of NoC complexity, depending on applications' reliability requirements. For example, the routing algorithm named Variant C is able to transmit packets as long as a fault-free path exists, with defect rates as high as 40%. Evolutions have also been studied, in order to improve the interconnect performances in the presence of a large number of faults.

Second, we propose a self-adaptive technique for the management of parallel applications, based on a fault-tolerant interconnect. The dynamic tasks mapping is based on the adaptive search of computing nodes, in order to reduce the application's energy consumption in the presence of variability.

Third, we present a high-level simulation model named VOCIS (*Versatile On-Chip Interconnect Simulator*), developed during this thesis. The model allows in-depth study of interconnection networks and fault-tolerant routings under complex settings, in order to meet the specific constraints of this work. The architecture and visualization features are described. Finally, we analyze and illustrate original experimental results obtained with this model.

***Keywords***—multi-cores, NoC, routing, fault tolerance, self-adaptivity

---

***Resumé***—La perspective de technologies nanométriques permet d'envisager l'avènement de processeurs constitués de centaines de cœurs de calcul. Néanmoins, l'utilisation de ces processeurs nécessitera de pallier aux problèmes de fiabilité et de variabilité inhérents à ces procédés de fabrication agressifs. Dans cette thèse, nous présentons un ensemble cohérent de techniques pour l'utilisation de processeurs multi-cœurs massivement parallèles, soumis à de forts taux de variabilité et de défaillance.

Tout d' abord, la fiabilité du réseau d'interconnexion est abordée, avec la présentation de plusieurs algorithmes de routage tolérants aux fautes, sans interblocages et sans table de routage pour une meilleure scalabilité. Les différentes variantes de ces algorithmes permettent d'ajuster la complexité du réseau sur puce, en fonction des besoins en fiabilité des applications. A titre d'exemple, le plus performant des algorithmes de routage peut acheminer les paquets tant qu'il existe un chemin sans défaillance, et ce jusqu'à 40% de ressources défectueuses. Plusieurs évolutions ont également été étudiées afin d'améliorer les performances du réseau en présence d'un nombre important de fautes.

Ensuite, nous proposons une technique auto-adaptative de gestion des applications parallèles. L'affectation dynamique des tâches se base sur la recherche adaptative des noeuds de calcul, afin de diminuer la consommation énergétique de l'application en présence de variabilité.

Enfin, nous présentons un modèle de simulation de haut-niveau appelé VOCIS (*Versatile On-Chip Interconnect Simulator*), développé pendant cette thèse. Il permet l'étude approfondie des réseaux d'interconnexion et des routages tolérants aux fautes dans des conditions complexes, afin de répondre aux contraintes propres à ce travail. Nous décrivons son architecture et ses capacités de visualisation. Finalement, nous analysons et illustrons plusieurs résultats originaux obtenus avec ce modèle.

***Mots clés***—multi-cœurs, réseau sur puce, routage, tolérance aux fautes, auto-adaptabilité

---

Laboratoire TIMA, 46, avenue Félix Viallet, 38031 Grenoble Cedex, FRANCE

ISBN 978-2-11-129182-9